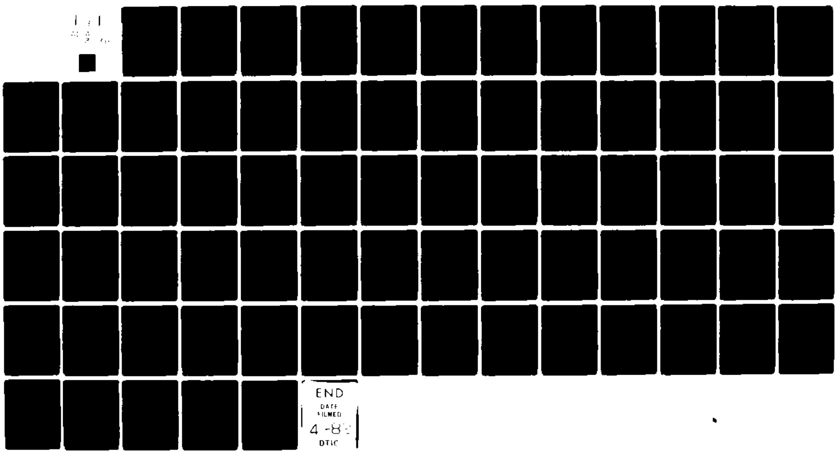
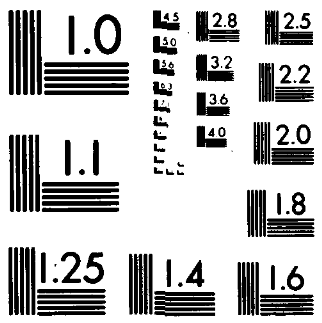




MICROCOPY RESOLUTION TEST CHART

ARO 15563.3-EL

(12)

AD A113238

# A STUDY OF DATA COMPRESSION AND EFFICIENT MEMORY UTILIZATION IN MULTIPROCESSOR AND DISTRIBUTED SYSTEMS

FINAL REPORT

C. V. Ramamoorthy

1 September 1978 - 31 August 1980

U. S. Army Research Office

Grant DAAG29-~~77~~-G-0189
78

Electronics Research Laboratory

University of California

Berkeley, California 94720

unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER | 2. GOVT ACCESSION NO. AD-A113238 | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle) A study of data compression and efficient memory utilization in multiprocessor and distributed systems | | 5. TYPE OF REPORT & PERIOD COVERED final 9/1/78 - 8/31/80 |
| | | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s) C. V. Ramamoorthy | | 8. CONTRACT OR GRANT NUMBER(s) DAAG29-78-G-0189 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS Electronics Research Laboratory University of California Berkeley, CA 94720 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office Post Office Box 12211 Research Triangle Park, NC 27709 | | 12. REPORT DATE |
| | | 13. NUMBER OF PAGES 71 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | | 15. SECURITY CLASS. (of this report) unclassified |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE NA |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

NA

18. SUPPLEMENTARY NOTES

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

scheduling, interleaved memory, data compression, Huffman co[illegible]g, data redundancy, pipelined processor.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Two organizations of interleaved memory system have been designed. The intelligent memory controller applies the scheduling algorithm to memory access requests. Three scheduling algorithms have been proposed and their performances are studied both theoretically and empirically. In data compression, the existing techniques have been classified into four areas. A multi-level compression scheme is proposed so that data is compressed through a set of cascaded stages.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

unclassified

## Abstract

Two organizations of interleaved memory system have been designed. The intelligent memory controller applies the scheduling algorithm to memory access requests. Three scheduling algorithms have been proposed and their performances are studied both theoretically and empirically. In data compression, the existing techniques have been classified into four areas. A multi-level compression scheme is proposed so that data is compressed through a set of cascaded stages.

Keywords: Scheduling, interleaved memory, data compression, Huffman coding, data redundancy, pipelined processor.

## 1. Introduction

In the past two years, we have studied the efficiency of memory utilization and data compression in multiprocessors and distributed systems. The work we have done can be summarized as follows: (1) design of an interleaved memory; (2) survey of data compression schemes and identification of future research directions.

The design of primary memory should consider (1) bandwidth; (2) response time; (3) size; (4) cost. The performance of final memory system can be evaluated by the above four parameters as evaluation criteria. From the study of memory access sequence of a pipelined processor, the access pattern is that instruction fetches are made in a sequence interlaced with operand accesses. The performance of the memory system may be improved by separating memory modules into two sets, one for instructions and one for data. In Section 2.7, the effects on memory performance due to separation and mergence of instruction and data modules are compared. We have studied two organizations of interleaved memories. The control of the memory system is the intelligent scheduler. The scheduler, using a scheduling algorithm, decides at the beginning of each memory subcycle whether to initiate a memory module and if so which module to initiate. The selection of which module to initiate is determined by the information about the requests in the associative buffers and by the knowledge about the status of the modules. Three scheduling algorithms are investigated in the design.

Data compression is any reversible encoding technique that produces a measurable reduction in the size of the data encoded.

By reversible, it is meant that the original data is recoverable from the compressed form. Due to the growth in the size of information processing, it is necessary to develop good data compression techniques which reduce the size of the stored information and the amount of internode communications.

We have surveyed data compression techniques and identify the problems for future research. A multi-leaved compression scheme is proposed so that data is compressed through a set of cascaded stages.

This report is divided into four sections. Section 2 presents the design of interleaved memory. Section 3 concentrates on data compression schemes. Lastly, Section 4 gives a conclusion of the report.

## 2. The Restricted Model - an Optimal Algorithm for Scheduling Requests on an Interleaved Memory System

### 2.1 Requirements for the Design of a Primary Memory

In a top-down design, the requirements and the attributes must first be identified before the system can be designed. Requirements are the constraints which the system must satisfy and they reflect the environment as well as the objectives of the system. Attributes, on the other hand, specify either options or evaluation criteria for qualitative comparisons of competitive systems that meet the system requirements. Attributes may be used to evaluate the tradeoffs in competing architectures and to obtain a feeling for the "goodness" of the architecture in realizing the system. The requirements for the design of a primary memory are:

(1) Bandwidth

The Bandwidth represents the average throughput of the memory system and is given in terms of bits returned/unit time. In a parallel memory system, the bandwidth is the sum of the bandwidths of all the modules (Bandwidth = $\sum_{module\ k}$ (word length of module k)*(average utilization of module k)/(cycle time of module k) where the average utilization of a module is the average fraction of time the module is busy. For the case of identical modules, the bandwidth can be written as:

$$\text{Bandwidth} = \frac{\begin{bmatrix}\text{number of}\\ \text{modules}\end{bmatrix} * \begin{bmatrix}\text{word}\\ \text{length}\end{bmatrix} * \begin{bmatrix}\text{average}\\ \text{utilization}\end{bmatrix}}{(\text{speed of module})}$$

$$\text{Bandwidth} = \frac{\text{constant} * \begin{bmatrix}\text{average number of}\\ \text{busy modules}\end{bmatrix}}{(\text{memory cycle time})} \qquad (2.1)$$

where the constant in Eq. 4.1 has a unit of (bits * memory cycle/unit time). The model of interleaved memories presented here assumes that all the modules are identical and the word lengths of each module are kept constant. The objective of maximizing the bandwidth is therefore equivalent to maximizing the average utilization of the modules.

(2) Response time

The response time is the delay between the time a request is accepted by the primary memory and the time the request is serviced, assuming that the datum resides in the primary memory. This is also called the waiting time of the requests.

(3) Size

This is the required memory size or capacity.

(4) Cost

This is the maximum allowable cost of the resultant design which satisfies the above requirements.

The design of the memory must satisfy the above requirements. Moreover, the performance of the final system can be evaluated by using these parameters as evaluation criteria.

## 2.2 Characteristics of the Access Sequence of a Pipelined Processor

In this section, we describe the characteristics of the access sequence of a pipelined processor. A pipelined organization in the most general sense, instead of specially structured pipelined computers with different arithmetic units (e.g., CRAY I), applications (e.g., vector processing), additional memory support (e.g., cache) and interconnections (e.g., ILLIAC IV), is assumed. The processor is further assumed to be executing directly from the main memory. The scheduling algorithms developed are general enough to be applicable to the interleaved memories of all the specially structured pipelined computers. However, the exact performance is not found for each type of machine.

A memory access sequence generated by a pipelined process has Class D dependencies as classified by Chang et. al. [CHA77]. A dependence is a logical relationship between two addresses such that the second address cannot be accessed (written or read) until the first has been accessed. Class D dependency is characterized by a machine with instruction level multiprogramming (from a large number of jobs), or a machine with sufficient lookahead or queueing hardware to allow dependencies to be bypassed. However, there still exist cases where the effects of dependencies cannot be eliminated. Anderson et. al. have identified three main sources of concurrency limitations which tend to reduce the performance of the pipe [AND67]. These are:

(a) Register interlock - When the current instruction needs a register modified by a previous instruction, the current instruction cannot be decoded until the previous instruction has finished.

(b) Branching - When a jump or a branch on condition instruction is encountered, further operations in the pipe cease until the target instruction has returned from the memory. Conditional branching poses an additional delay because the branch decision depends on the outcome of arithmetic operations in the execution units.

(c) Interrupts - When an interrupt occurs in the pipe, it is necessary to sequentialize the execution of instructions in the pipe in order to determine the exact source of the interrupt. This sequentialism in execution would degrade the performance of the pipe.

Various methods have been introduced to solve these dependency problems [TOM67]. For example, register interlocks can be solved by using forwarding; the sequentialism due to interrupts can be eliminated by using imprecise interrupts as in IBM 360/91. The most predominant effect on the performance of the memory is due to branching. When a branch or a conditional branch instruction is encountered, request supply to the memory discontinues until the condition code has been set and the target instruction has returned from the memory. The utilization of the memory therefore decreases. The effects on the memory performance due to branching dependencies are studied in section 2.8.

In addition to the effects due to address dependencies, the order in which instructions and data are requested also affects the memory performance. For a pipelined processor, the request stream is a sequence of instruction-operand fetch pairs. However, not every instruction involves an operand fetch and if the bus is wide enough, two or more instructions can be fetched in one access. A notable characteristic in this access pattern is that instruction fetches are made in a sequence interlaced with operand accesses. The performance of the memory system may be improved by separating the memory modules into two sets, one for instructions and one for data. In section 2.7, the effects on memory performance due to separation and mergence of instruction and data modules are compared.

## 2.3 Previous Work on the Study of Interleaved Memories

One of the early successful implementations of interleaved memories is in the IBM 360/91 [BOL67]. In this computer, the storage system is made up of an interleaved set of memory modules and the degree of interleaving equals the number of memory modules. The memory can service a string of sequential requests by starting, or selecting, a storage unit every cycle until all are busy. In effect, the storage cycles are all staggered (see Fig. 2.2). By using a set of buffers called the request stack, conflicting requests which access the same module can be resolved by allowing only one of these requests to access the module and storing the rest in the request stack to be issued in later cycles. Simulation results were shown for the average access time and the bandwidth with various degrees of interleaving.

The earliest attempt to model the performace of interleaved memories was done by Hellerman [HEL67]. By assuming a saturated request queue (a queue in which requests are never exhausted) with random requests, and if no provision is made for the queueing of the requests on busy modules, the request queue is scanned until a repeated request is found. This constitutes a collision. Hellerman's results show that with m memory modules, the average number of requests scanned before a collision is approximately $m^{0.56}$ for $m$ between 1 and 45. This is taken to be an indication of bandwidth. Knuth and Rao [KNU75] show an alternate exact way to calculate the bandwidth. However, both of these results are pessimistic because they do not allow the queueing of conflicting requests to the same module and the randomness assumption is not tenable in real programs.

Burnett et. al. have developed a number of models on parallel memories. In two of these models [BUR70, BUR73], they assume that the modules operate synchronously (all modules start and end their cycles simultaneously) and a scanner scans a saturated request queue and admits new requests to service until it attempts to assign a request to a busy module. In two other models, [BUR75], they further assume that a set of blockage buffers is present so that requests made to a busy module can be stored and issued in later cycles. The scanner continues to scan the request queue until all the modules have been allocated or all the buffers are occupied. In effect, the maximum size of the request queue inspected by the scanner never exceeds b+m where b is the number of buffers and m is the number of memory modules. They have also studied a request model similar to Strecker's model [STR70] by assuming a probability $\alpha$ for the succeeding request to request the next module in sequence and a probability of $(1-\alpha)/(m-1)$ to request any other module. They have developed two algorithms that modified the request pattern in order to increase the bandwidth. The first one is called the Instruction-Data Cycle Structure, which distinguishes the request queues into sub-queues, the instruction queue and the data queue. These two sub-queues are inspected in alternate memory cycles. They found that there are improvements from -4% to 12% in bandwidth (the number of modules varies from 8 to 16) over a model with four blockage buffers and a single queue [BUR75]. The second algorithm, the Group Request Structure, separates a memory cycle into two sub-cycles; the first sub-cycle is used for servicing the instruction queue, and the second sub-cycle is used for servicing the data queue. They found that there are 8% to 16% improvements over the same Instruction-Data Cycle Structure algorithm and found that the theoretical predictions of Burnett and Coffman fit well with the simulation results for the fetching of instructions, but their predictions do not fit well with the simulation results for data requests which are more random than instruction requests and are difficult to model accurately.

Many other researchers have studied models of parallel memories. These include Flores [ FLO64 ] , Skinner and Asher [ SKI69 ], Ravi [ RAV72], Sastry and Kain [ SAS75 ], Briggs and Davidson [ BRI77 ] , Chang, Kuck and Lawrie [ CHA77 ] , Smith [ SMI77 ] and Hoogendoorn [ HOO77 ]. These studies are directed toward multi-processor systems and we will not describe them here.

In the remainder of this section, the deficiencies found in the previous models are summarized.

(1) All the previous models assume that the memories operate synchronously. As Burnett and Coffman pointed out, simultaneous memory operations offer more opportunity to take advantage of program behavior in a particular memory system [BUR75]. However, with synchronous operations, there is the problem of returning the results of the accesses from the memory. Since the results from each module are available simultaneously, extra data paths or queues are needed to return these data to the processor. Further, a pipelined processor

usually makes requests in sequences rather than in batches. Therefore it is desirable to study a model in which the memory modules operate out of phase. By out of phase, we mean either a) the initiations of the modules are asynchronous or b) the initiations of the modules are timed by a clock and during a clock interval, at most one module can be initiated. Because the operations of asynchronous modules are much more difficult to control, only case (b) is considered in this design.

(2) Very few studies have been made to minimize the waiting time of a request to the memory. Flores [FLO64] has made a quantitative study relating the waiting time factor to the memory cycle time, the input/output time and the worst case execution time for different numbers of memory banks. However, his studies were directed toward the effect of interference from the input/output units and there was no queueing of requests. In other models, a saturated request queue is assumed, and the effects of waiting time are not considered. When the queue size is finite, it is possible to develop algorithms which optimize for the amount of waiting time in the queue, e.g., minimize the average waiting time of requests in the queue. In this section, the amount of queued requests is assumed to be finite so that the effects of waiting time can be studied.

(3) None of the previous work considers the effects of dependencies on the memory performance. Request supply to the memory ceases when a dependent instruction is executed until the dependency has been resolved. The effects of dependencies are difficult to determine because they vary strongly with the configuration of the pipe and the strategies employed in the pipe to resolve them. Request rate to the memory may also decrease for other reasons. For example, in the IBM 360/91, there is a

small amount of instruction buffers in the CPU which serve as another level of the memory hierarchy. When a small loop occurs such that all the instructions of the loop fit in the instruction buffers, instruction accesses to the memory stop until execution of the loop is finished. Other machines may have different approaches. However, the evaluation of memory performance for a specific machine is too restrictive. We take an approach which first evaluates the performance for the general case of an interleaved memory with a saturated, non-dependent request stream. The degradation in performance due to dependence in the requests is then estimated subsequently.

## 2.4 The Organization of Primary Memory for a Pipelined Processor

We present in this section two different implementation alternatives of interleaved memories (Organization I and Organization II). The two organizations differ in the configurations of the request buffers. In Organization I, a single set of request buffers is assumed to be shared by all the modules and in Organization II, individual request buffers exist for each module. The general assumptions made are as follows:

(1) The request rate from the processor is assumed to be high enough so that any empty buffer in the memory system is filled up by an incoming request immediately. Buffers are assumed to exist at the processor end so that any additional requests generated by the processor can be queued there. The requests that can be served by the modules are those that exist in the buffers only. This assumption is made because we want to get an upper bound on the performance of the memory. In a practical system, the memory is usually the bottleneck and our assumption is therefore valid.

(2) Each request is assumed to be an integer from 0 to m-1, which is the module it requests, and is obtained as the residue of dividing the address by m.

(3) The service time of each module (the read time or the write time) for a request is assumed to be constant. This is a good model for semiconductor memories. We also assume that a memory module, once initiated to start a memory cycle, is not available until the end of the cycle.

(4) A memory cycle time is the time it takes for a memory module to service a request. Each memory cycle is assumed to consist of m equally spaced memory sub-cycles. It is further assumed that exactly one module can be initiated to service a request at the beginning of a memory sub-cycle and it takes m sub-cycles (1 memory cycle) to service the request for all the modules, i.e., homogeneous service times. With this assumption, the problem of multiple data paths is resolved because at most one module finishes in each sub-cycle and the system is never confronted with returning results from more than one module simultaneously. The modules are therefore clocked by the memory sub-cycles.

In Organization I (Fig. 2.1), there are m memory modules; a single set of b+1 associative buffers, $B_T$, $B_1$, $B_2$,...$B_b$; and an intelligent scheduler which schedules a memory module to start a memory cycle. The modules operate out of phase in a fashion called staggered cycles. One example of a staggered cycle is shown in Fig. 2.2. The set of b+1 associative buffers are used to store incoming requests. A request queued on a specific module can be retrieved in one associative search operation. Whenever a request is taken out from a buffer, all the requests behind it are pushed one location up so that $B_T$ is empty. The buffer $B_T$ has an additional function, namely, to receive requests from the bus. Due to our assumption of high request rate, $B_T$ is filled immediately whenever it is empty. The queueing discipline for the requests in the buffers directed towards the same module is essentially First-In-First-Out (FIFO). Other queueing disciplines are not studied because only uni-processor systems are considered in this design.

The center of the control in the memory system is the intelligent scheduler. The scheduler, using a scheduling algorithm, decides at the beginning of each memory sub-cycle whether to initiate a memory module and if so which module to initiate. The selection of which module to initiate is determined by the information about the requests in the associative buffers and by the knowledge about the status of the modules (free or busy). Three scheduling algorithms are investigated in this design.

(1) Algorithm 2.1 Round-Robin (RR)

All the modules are initiated in a round-robin fashion regardless of whether a request is queued on the module. The scheduler does not make use of any information about the status of the system. The implementation of this algorithm is very simple and the scheduler only has to know the current module initiated. In Fig.    , the Gantt Chart for the operation of a 4-way interleaved memory with RR scheduling algorithm is shown. This is the scheduling algorithm that is implemented in more interleaved memory systems today.

(2) Algorithm 2.2 First-Free-First (FFF)

In this algorithm, only the information about the status of the modules (free or busy) is utilized by the scheduler. There is a FIFO list of free modules. At the beginning of a memory sub-cycle, the scheduler puts a busy module to the end of the free list if this module finishes its cycle. It will then initiate the module at the head of the free list if there are any requests queued on it, otherwise the module at the head is appended to the tail of the free list and no other modules are checked in this cycle. The scheduler may also check all the subsequent modules in the free list, but the time for this is proportional to the number of modules and is not feasible when this number is large.

(3) Algorithm 2.3 Maximum-Work-Free-Module-First (MWFMF)

In this algorithm, both the information about the status of the modules and the requests in the buffers are utilized by the scheduler. There is a dynamic list of free modules. Conceptually, at the beginning of a memory sub-cycle, the buffers are checked associatively to see if any requests are queued on the free modules. If there is none, no module is initiated. If at least one exists, an associative search is made on the buffers and the module with the maximum number of requests queued on it is initiated. In case of ties, only the first one is initiated (fig. 2.3a). The implementation of this algorithm can be done by using an additional associative memory of size m in the scheduler (Fig. 2.3b). Each word in this associative memory can function as a counter and is used to indicate the number of requests queued on the corresponding module. The corresponding word is incremented/decremented when a request enters/leaves the request buffers. The free module with the maximum number of requests can be obtained by performing a maximum search on those words in this associative memory corresponding to the free modules, e.g., [RAM78a] (see the associative memory design in Chapter 5). The maximum search algorithm shown in [RAM78a] is parallel by word and serial by bit and the time to perform a maximum search is proportional to the number of bits in the memory. The speed of this algorithm is therefore proportional to $\lceil \log_2 b \rceil$.

In addition to the overhead related to the execution of the scheduling algorithm, there is also the overhead of selecting the request from the associative buffers and sending it to the memory module. This overhead consists of matching the selected module number against all the requests in the buffers and selecting the first request if multiple responses occur in the match. Using a bit-serial word-parallel equality matching algorithm, e.g. [FOS68], this overhead is proportional to $\lceil \log_2 m \rceil$. In general, the overheads associated with the three scheduling algorithms are very small, and the selection of a module and the corresponding request to be initiated in the next sub-cycle can be overlapped with the current sub-cycle.

At the end of each memory sub-cycle, at most one request is serviced. The result is sent back to the processor. The necessary queue for storing these results is excluded from the memory model.

The requests of the system come into the memory in a specific pattern. Two types of access patterns are considered in this design:

(1) Random accesses with no address dependency. All the addresses have no correlation and are independent of each other. This can be used to model the request stream from computer systems with instruction level multiprogramming or multi-processor systems where the number of processors is larger than the number of modules.

(2) Accesses from the execution trace of a monoprogrammed pipelined computer:

The addresses in the execution traces are correlated and they represent a similar addressing behavior when the actual program is executed on a pipelined processor. We have used execution traces from a pipelined processor, representing large scientific applications, the CDC 7600, in this study.

Organization II is similar to Organization I except that separate sets of buffers exist for each module (Fig. 2.4). Requests from the processor are continuously moved into the buffers of each module via $B_T$ until a request in $B_T$ is directed toward a module whose buffers are already full. The request in $B_T$ is blocked, and as a result, further requests are blocked from entering the memory. When the module responsible for this blocking has finished servicing its current request, one request from its buffers is serviced which results in an empty buffer. The blocking request in $B_T$ is moved into this empty buffer. Because of the independent queues, one or more requests can then be accepted to the memory system until the previous blocking situation occurs with one of the modules. When b=0, there is only one buffer, $B_T$, in the system and this is exactly the same as Organization I when b=0. The buffers used in this organization are simpler than those of Organization I. Associative search capabilities are not necessary for these buffers. The implementation of the scheduler is similar to that of Organization I. The advantage with this system is that the request buffers are simple shift registers and therefore are cheaper. However, in order for this organization to operate at full capacity, more than one request may have to be moved across the bus into the memory in a memory sub-cycle. As we recall, we assumed that a pipelined processor generates in the order of one request every memory sub-cycle; therefore, the blocking situation may not always occur and the buffers are under-utilized. Further, it is necessary to build a faster bus so that multiple requests can be moved across the bus in a memory sub-cycle. We can assume that sufficient requests are queued in the processor so that the need of moving more than one request into the memory system during a sub-cycle can be satisfied. An alternative is to allow a maximum of one request to be accepted in every sub-cycle. This results in a degraded performance for Organization II because the system is not operating with the maximum request rate.

Since the two organizations discussed are operating in steady state and the systems discussed are balanced, the average arrival rate and the average waiting time are related by Little's Formula.

Let

$e_B$ = utilization of the buffers $B_1, \ldots, B_b$
(=1 for Organization I)

$e_T$ = utilization of buffer $B_T$
(=1 for both organizations)

B = number of buffers in $B_1, \ldots, B_b$
(=b for Organication I; =m*b for Organization II)

$u_{m,b}$ = expected utilization of the modules

$w_{m,b}$ = expected waiting cycles of the requests

M = expected number in the system

$\lambda$ = expected arrival rate

W = expected waiting time of the requests

Then

$$M = (e_B{}^*B + 1) + u_{m,b}{}^*m \qquad (4.2)$$

$$\lambda = u_{m,b} \qquad (4.3)$$

$$W = m^*w_{m,b} \qquad (4.4)$$

and they satisfy Little's Formula,

$$M = \lambda^*W$$

Eq. (4.3) is true because in a balanced system the expected arrival rate equals the expected service rate. The physical importance of Little's Formula lies in the fact that the average utilization and the average number of waiting cycles are related. Once one of them is obtained, the other can be calculated easily. Further, it also shows that Organizations I and II are equivalent as far as the average behavior is concerned. The only difference lies in the buffer utilization which is less than 1 in Organization II whereas the buffers are fully utilized in Organization I. In the next section, we present our evaluations for Organization I only because the two organizations are equivalent and the results are directly applicable. It is shown that the MWFMF algorithm minimizes the average completion time of the requests. This result only demonstrates that the MWFMF algorithm is superior, but the exact throughput values of the system cannot be obtained analytically. The techniques that are used to evaluate the performance of these two organizations are embedded Markov analysis with random requests and simulations with random requests and execution traces and they are shown in Section 2.6.

## 2.5 Optimality of the MWFMF Scheduling Algorithm

In proving the optimality, it is assumed that the requests in the request queue are independent, randomly generated and of a finite size. The size of the associative buffers may be greater than, equal to, or less than the number of requests in the request queue. In a pipelined processor, memory requests can be generated continuously until a dependency occurs. At this point, the request stream is discontinued until the dependency has been resolved. Because of the high request rate assumption, the requests generated between two dependencies can be assumed to exist in the reqest queue after the first dependency has been resolved. However, in a practical implementation, the pipelined processor is able to look ahead only a fixed amount of instructions and this is modeled by a fixed and finite amount of associative buffers in the system (which may be greater than, less than or equal to the size of the request queue). The intelligent scheduler is allowed to examine the associative buffers in making the scheduling decision. The objective of the scheduling algorithm is to complete the service of the requests in the request queue as fas as possible so that the throughput of the memory is maximized. The symbols used in the following theorems are:

$b$ = number of associative buffers - 1;

$m$ = number of memory modules;

$N$ = total number of requests that have to be serviced between two dependencies;

$\{(\ell_1,i_1),(\ell_2,i_2),\ldots,(\ell_m,i_m)\}_k$ = state of the memory system, where

$(\ell_j,i_j)$ = state of module $j$;

$\ell_j$ = number of requests queued on module $j$ in the buffers;

$$\sum_{j=1}^{m} \ell_j = b + 1 \text{ and } \ell_j \geq 0 \quad j = 1, 2, \ldots, m$$

$$i_j = \begin{cases} 0 & \text{if module } j \text{ is free} \\ n & 0<n<m \text{ if module } j \text{ is busy} \end{cases}$$

In the case that module $j$ is busy, $n$ is the number of cycles that module $j$ has serviced its current request. The number of cycles remaining before the completion of service for the current request is (m-n) mod m.

k = variable used in the induction proof indicating the numb r of remaining requests to be serviced (not including those in the associative buffers);

$C_{max}\{(\ell_1,i_1),(\ell_2,i_2),\ldots,(\ell_m,i_m)\}_k$ = maximum completion time for the state;

$EC_{max}\{(\ell_1,i_1),(\ell_2,i_2),\ldots,(\ell_m,i_m)\}_k$ = expected maximum completion time for the state.

Before the main theorem can be stated, the following three lemmas must first be proved. Lemma 2.1 establishes the need for executing the MWFMF scheduling algorithm at the beginning of each sub-cycle. Lemma 2.2 establishes a basis for the induction proof of the main theorem and it also shows the optimality of the MWFMF algorithm when the buffer size is very large so that all the requests in the request queue reside in the buffers. Lemma 2.3 augments Lemma 2.2 by further showing that algorithm MWFMF minimizes the sum of completion times of all the requests.

LEMMA 2.1

(1) In a period of m sub-cycles, every module can be initiated at most once.

(2) At the beginning of each sub-cycle, at least one free module is available for scheduling.

Proof

(1) Obvious, because each module takes a time of m sub-cycles to service a request.

(2) Consider a time interval of m sub-cycles. Since at most one module can be scheduled in each sub-cycle, the total number of modules scheduled in m sub-cycles is less than or equal to m. At the beginning of its current sub-cycle, if a module is scheduled m sub-cycles ago, then it will finish its service at the current sub-cycle and is available for scheduling. If a module is not scheduled m sub-cycles ago, then the total number of modules scheduled in the last m sub-cycles is less than m. Therefore, at least one module is available for scheduling at the beginning of a sub-cycle.

Q.E.D.

LEMMA 2.2

If all the requests in the request queue reside in the associative buffers (that is, the buffers are large enough to accompany all these requests), then algorithm MWFMF minimizes the maximum completion time for independent, random requests in Organization I.

Proof

The maximum completion time is governed by the longest queue in the system.

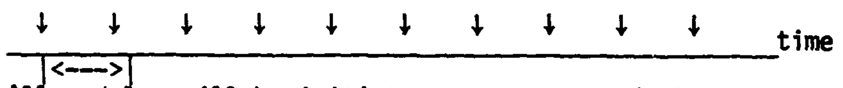
Assume without loss of generality:

$\ell_1 > \ell_2 \cdot \cdot \cdot > \ell_m$

Case 1: $i_1 = 0$,

MWFMF schedules module 1 first.

initiate module 1



All modules will be initiated at most once in here due to lemma 4.1 (if number queued on it is non-zero) and all requests queued every module except 1 can be initiated before the last request queued on module 1 is initiated.

$C_{max} = \ell_1 * m + 1$ sub-cycles (initiate module $j \neq 1$ first)

Case 2: $i_1 > 0$

Let module $j$ be the module such that

$i_j = 0$ and $i_1 > 0,\ i_2 > 0, \ldots, i_j > 0$.

That is, module $j$ is the free module with the largest amount of queued requests. This will be the module scheduled by the algorithm MWFMF. In fact, the module scheduled at this point is unimportant because the maximum completion time is governed by module 1.

$C_{max} = \ell_1 * m + (m - i_1)$ sub-cycles

Therefore:

min $C_{max} = \ell_1 * m + (m - i_1) \bmod m$ sub-cycles

Optimum algorithm: MWFMF

On the other hand, if $\ell_1 = \ell_2 > \ldots > \ell_m$ and $i_1$, $i_2 = 0$, then the $C_{max}$'s are identical whether module 1 or 2 is scheduled first. A similar proof holds for the case $\ell_1 \geq \ell_2 \geq \cdot \cdot \cdot \geq \ell_m$.

LEMMA 2.3

If all the requests in the request queue reside in the associative buffers, then algorithm MWFMF minimizes $\sum C_j$ for independent, random requests in Organization I where $C_j$ is the completion time for the $j$'th request.

<u>Proof</u>

Assume without loss of generality:

$$\ell_1 > \ell_2 > \cdot \cdot \cdot > \ell_m$$

Consider two modules a, b, such that $i_a = 0$, $i_b = 0$ and $\ell_a > \ell_b$. Let $C_{a,b}(C_{b,a})$ be the sum of completion times of scheduling a before b (b before a) for modules a and b only. If b is scheduled before a, then

$$C_{b,a} = C_b + C_a = \frac{m}{2}[(\ell_a + 1)\ell_a + (\ell_b + 1)\ell_b] + \ell_a$$

Comparing this with the case of scheduling a first, it is found that:

$$C_{a,b} = C_a + C_b = \frac{m}{2}[(\ell_a + 1)\ell_a + (\ell_b + 1)\ell_b] + \ell_b$$

Since $\ell_a > \ell_b \Rightarrow C_{a,b} < C_{b,a}$, this implies that scheduling the module with a larger amount of queued requests can reduce $\sum C_j$. By adjacent pairwise interchange, it is therefore better to schedule the module with the maximum amount of queued requests if it is free. If the module is not available, scheduling the free module with the maximum amount of queued requests is also optimum.

Q.E.D.

From the proofs of Lemmas 2.2 and 2.3, it is seen by using the MWFMF algorithm that,

(1) The throughput of the memory is at a maximum because the maximum time to complete a set of jobs is minimized (Lemma 2.2).

(2) The average waiting time is minimized. This is because $C_j$, the completion time for the $j$'th job, equals the waiting time for the $j$'th job.

$W_j = C_j - 0$, (all the jobs are available at t = 0). As a result, average waiting time = $\sum W_j/M$ is also minimized (Lemma 2.3).

THEOREM 2.6

If all the requests in the request queue do not reside in the associative buffers, (that is, the buffers are not large enough to accompany all the requests in the request queue), then algorithm MWFMF minimizes the expected maximum completion time for independent, random requests in Organization I.

Proof

In order to prove this theorem, the following two parts must be proven and the theorem follows from the result of part (a).

(a) Algorithm MWFMF minimizes the expected maximum completion time for independent, random requests.

(b) Let states

$S_1 = \{\ldots, (\ell_a^1, i_a), (\ell_b^1, i_b), \cdots\}_k$

$S_2 = \{\ldots, (\ell_a^2, i_a), (\ell_b^2\ i_b), \cdots\}_k$

where "..." indicates that the remaining states are identical for $S_1$ and $S_2$.

Since the states of other modules are identical, and we assume that

$\ell_a^2 > \ell_a^1$;

$\ell_b^1 > \ell_b^2$;

$\ell_a^1 + \ell_b^1 = \ell_a^2 + \ell_b^2$;

and

$m \geqq i_a > i_b > 0$ or $m \geqq i_b > i_a > 0$ with equal probability.

If $\ell_a^2 > \ell_b^1$, then $EC_{max}(S_1)_k \leqq EC_{max}(S_2)_k$;

If $\ell_a^2 = \ell_b^1$, then $EC_{max}(S_1)_k = EC_{max}(S_2)_k$.

These two parts can be proved by induction. The truth is first established for k = 0, i.e., when all the requests reside in the buffers. These parts are then assumed to be true for any positive integer k and the proof is complete by proving the case of k + 1.

(1) k = 0

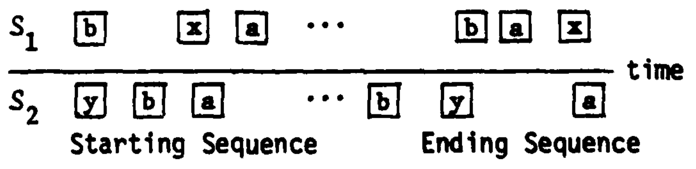
(a) MWFMF is optimal. This is established by Lemma 2.2.

(b) If there exists module z such that $\ell_z > \ell_a^2$, and since $\ell_a^2 \geqq \ell_b^1 > \ell_b^2$ and $\ell_a^2 > \ell_a^1$, then the maximum completion time for both $S_1$ and $S_2$ depends on $\ell_z$ and are identical. Therefore,

$EC_{max}(S_1)_0 = EC_{max}(S_2)_0$

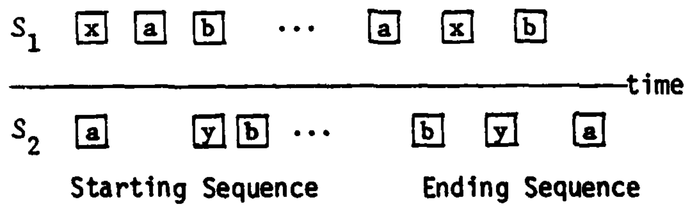
If there does not exist module z such that $\ell_z > \ell_a^2$, then the maximum completion time of $S_2$ depends on module a. Let there be two modules, x in $S_1$ and y in $S_2$ such that $\ell_a^2 > \ell_x^1 > \ell_a^1$, $\ell_b^1 > \ell_y^2 > \ell_b^2$ and $i_x = i_y = 0$. The following three cases can be identified.

(1) $\ell_a^2 > \ell_b^1$, $i_b = 0$, $i_a < m$



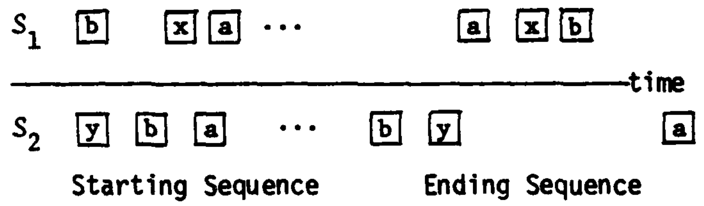
$C_{max}(S_1)_0 = EC_{max}(S_1)_0 < C_{max}(S_2)_0 = EC_{max}(S_2)_0$

(2) $\ell_a^2 > \ell_b^1$, $i_a = 0$, $i_b$

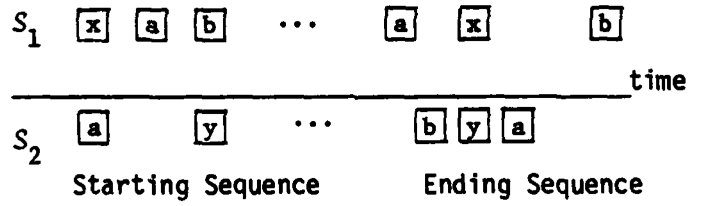


$$C_{max}(S_1)_0 = EC_{max}(S_1)_0 < C_{max}(S_2)_0 = EC_{max}(S_2)_0$$

(3) $\ell_a^2 = \ell_b^1$, $i_b = 0$, $i_a < m$



$\ell_a^2 = {}_b^1$, $i_a = 0$, $i_b < m$



Since $\ell_a^2 = \ell_b^1$, this implies that $\ell_a^1 = \ell_b^2$, therefore the states $S_1$ and $S_2$ are symmetric in the states of the modules a and b and the probability that $i_b = 0$, $i_a < m$ is equally likely as the probability that $i_a = 0$, $i_b < m$.

$$EC_{max}(S_1)_0 = C_{max}(S_1 | \, i_b = 0, i_a < m)_0 * Pr(i_b = 0, i_a < m)$$

$+ C_{max}(S_2 \mid i_a = 0, i_b < m)_0 * Pr(i_a = 0, i_b < m)$

$= EC_{max}(S_2)_0$

(II) Induction hypothesis:

Assume that the theorem is true for a positive integer k, that is,

(a) MWFMF algorithm minimizes the expected maximum completion time for independent, random requests when the number of remaining requests in the request queue is k.

(b) If $\ell_a^2 > \ell_b^1$, then $EC_{max}(S_1)_k \leqq EC_{max}(S_2)_k$;

If $\ell_a^2 = \ell_b^1$, then $EC_{max}(S_1)_k = EC_{max}(S_2)_k$.

(III) When the number of remaining inputs is k+1,

(a) Without loss of generality, let modules 1, 2, ..., $j$ be the set of free modules. Choose any two modules, say 1 and 2, so that $\ell_1 > \ell_2$ and there does not exist $p \in \{1,2,\ldots,j\}$ such that $\ell_1 > \ell_p > \ell_2$. We want to compare the difference between scheduling module 1 and module 2.

(1) Schedule module 1 in this sub-cycle,

$\{(\ell_1,0), (\ell_2,0), \ldots, (\ell_m, i_m)\}_{k+1}$

A new input now enters the buffers; this input can be a request directed to any module in the set with equal probability 1/m (due to the assumption of independent, random requests).

New states after scheduling module 1:

1 enters: $S^1 = \{(\ell_1,1), (\ell_2,0), \ldots, (\ell_m, (i_m+1) \bmod m)\}_k$

2 enters: $S^2 = \{(\ell_1-1, 1),(\ell_2+1,0), \ldots,(\ell_m,(i_m+1) \bmod m)\}_k$

- - -

m enters: $S^m = \{(\ell_1-1, 1), (\ell_2, 0), \ldots,(\ell_m+1,(i_m+1) \bmod m)\}_k$

(2) Schedule module 2 in this sub-cycle.

$\{(\ell_1, 0), (\ell_2,0), \ldots, (\ell_m,i_m)\}_{k+1}$
$\Rightarrow \{(\ell_1,0),(\ell_2-1,1), \ldots,(\ell_m,(i_m+1) \bmod m)\}_{k+1}$

New states after scheduling module 2:

1 enters: $\bar{S}^1 = \{(\ell_1+1, 0), (\ell_2-1, 1 , \ldots,(\ell_m,(i_m+1) \bmod m)\}_k$

2 enters: $\bar{S}^2 = \{(\ell_1,0),(\ell_2,1), \ldots,(\ell_m, (i_m+1) \bmod m)\}_k$

- - -

m enters: $\bar{S}^m = \{(\ell_1,0),(\ell_2-1, 1), \ldots,(\ell_m+1,(i_m+1) \bmod m\}_k$

It is seen that $EC_{max}(S^1)<EC_{max}(\bar{S}^2)$, $EC_{max}(S\ )<EC_{max}(\bar{S}^1)$ and $EC_{max}(S^j)<EC_{max}(\bar{S}^j)$ for $j\neq 1,2$. In proving $EC_{max}(S^1)<EC_{max}(\bar{S}^2)$, we can use the induction hypothesis II(b) and let $i_a=0$, $i_b=1$, $\ell_a^1=\ell_1$, $\ell_a^2=\ell_1$ $\ell_b^2=\ell_2$. The other parts can similarly be proved. Since the expected $C_{max}$ is a weighted sum of the expected $C_{max}$ of all the corresponding states, it is therefore better to schedule module 1, the module with a longer queue, first. By using the adjacent pair-wise interchange argument, the free module with the maximum number of queued requests should be scheduled first.

(b) In proving this theorem, the following parts are identified.

(1) $\ell_a^2>\ell_b^1$; both modules a and b are not scheduled in the current sub-cycle. This can be due to (1) $i_a>0$ and $i_b>0$; i.e., both modules are busy; or (2) there exists a free module z such that $\ell_2$ is greater than $\ell_a^2$ if $i_a=0$ or $\ell_b^1$ if $i_b=0$. Since it is assumed in the induction

hypothesis II(a) that free modules with a longer queue should be scheduled, therefore module z will be scheduled in this case.

After module z is scheduled, a new input enters the buffers.

For state $S_1$ $\{\ldots, (\ell_a^1, i_a), (\ell_b^1, i_b), \cdots\}_{k+1}$

a enters:

$$S_1^a = \{\ldots, (\ell_a^1+1, (i_a+1) \mathit{mod}\ m), (\ell_b^1, (i_b+1) \mathit{mod}\ m), \cdots\}_k$$

b enters:

$$S_1^b = \{\ldots, (\ell_a^1, (i_a+1) \mathit{mod}\ m), (\ell_b^1+1, (i_b+1) \mathit{mod}\ m), \cdots\}_k$$

j,j≠a,b enters:

$$S_1^j = \{\ldots, (\ell_a^1, (i_a+1) \mathit{mod}\ m), (\ell_b^1, (i_b+1) \mathit{mod}\ m), \cdots\}_k$$

For state $S_2$ $\{\ldots, (\ell_a^2, i_a), (\ell_b^2, i_b), \cdots\}_{k+1}$

a enters:

$$S_2^a = \{\ldots, (\ell_a^2+1, (i_a+1) \mathit{mod}\ m), (\ell_b^2, (i_b+1) \mathit{mod}\ m), \cdots\}_k$$

b enters:

$$S_2^b = \{\ldots, (\ell_a^2, (i_a+1) \mathit{mod}\ m), (\ell_b^2+1, (i_b+1) \mathit{mod}\ m), \cdots\}_k$$

j,j≠a,b enters:

$$S_2^j = \{\ldots, (\ell_a^2, (i_a+1) \mathit{mod}\ m), (\ell_b^2, (i_b+1) \mathit{mod}\ m), \cdots\}_k$$

By the induction hypothesis,

$$EC_{max}(S_1^a)_k < EC_{max}(S_2^a)_k$$

$$EC_{max}(S_1^b)_k < EC_{max}(S_2^b)_k \qquad \text{if } \ell_a^2 > \ell_b^1 + 1$$

$$EC_{max}(S_1^b)_k = EC_{max}(S_2^b)_k \qquad \text{if } \ell_a^2 = \ell_b^1 + 1$$

$$EC_{max}(S_1^j)_k \quad EC_{max}(S_2^j)_k \qquad \forall\ j \neq a,b$$

Therefore

$$EC_{max}(S_1)_{k+1} \leq EC_{max}(S_2)_{k+1}$$

(2) $\ell_a^2 > \ell_b^1$ and there exists a module x such that $i_x = 0$

and $\ell_a^2 > \ell_x > \ell_a^2$ if $i_a = 0$ or

$\ell_b^1 > \ell_x > \ell_b^2$ $i_b = 0$

Let us look at the first case:

$$S_1 = \{\ldots,(\ell_a^1,0),(\ell_b^1,i_b),\ldots,(\ell_x,0),\cdots\}_{k+1}$$

$$S_2 = \{\ldots,(\ell_a^2,0),(\ell_b^2,i_b),\ldots,(\ell_x,0),\cdots\}_{k+1}$$

According to the MWFMF algorithm, module x should be scheduled in $S_1$ and module a should be scheduled in $S_2$. It is necessary to compare the expected $C_{max}$ after these have been scheduled. Suppose module x is not scheduled in both states, from part III(b)(1), it is seen that $EC_{max}(S_1|a\ scheduled)_k < EC_{max}(S_2|a\ scheduled)_k$. However, due to the induction hypothesis, II(a), scheduling x in state $S_1$ would be better than scheduling a because $\ell_x > \ell_a^1$.

$$EC_{max}(S_1|x\ scheduled)_k < EC_{max}(S_1|\ a\ scheduled)_k$$

Therefore:

$$EC_{max}(S_1|\ x\ scheduled)_k < EC_{max}(S_2|\ a\ scheduled)_k$$

and

$$EC_{max}(S_1)_{k+1} < EC_{max}(S_2)_{k+1}$$

The other case, i.e., $\ell_b^1 > \ell_x > \ell_b^2$ and $i_b=0$ can be similarly proved. For the remainder of the proof of this theorem, it is assumed that $\ell_a^2 \geq \ell_b^1 > \ell_x$, for all $x \neq a,b$ and $i_x=0$.

(3) $\ell_a^2 > \ell_b^1,\ 0 < i_a < m,\ i_b = 0$

Due to the induction hypothesis, module b should be scheduled in $S_1$ and $S_2$.

For the state $S_1$, schedule module b in this sub-cycle,

$$\{\ldots, (\ell_a^1, i_a), (\ell_b^1, 0), \cdots\}_{k+1}$$

$$\Rightarrow \{\ldots, \ell_a^1, (i_a+1) \bmod m), (\ell_b^1-1, 1), \cdots\}_{k+1}$$

New input enters the buffer:

a enters: $S_1^a = \{\ldots, (\ell_a^1+1, (i_a+1) \bmod m), (\ell_b^1-1, 1), \cdots\}_k$

b enters: $S_1^b = \{\ldots, (\ell_a^1, (i_a+1) \bmod m), (\ell_b^1, 1), \cdots\}_k$

j, j≠a,b enters:

$$S_1^j = \{\ldots, (\ell_a^1, (i_a+1) \bmod m), (\ell_b^1-1, 1), \cdots\}_k$$

For state $S_2$, schedule module b in this sub-cycle:

$$\{\ldots, (\ell_a^2, i_a), (\ell_b^2, 0), \cdots\}_k$$

$$\Rightarrow \{\ldots, (\ell_a^2, (i_a+1) \bmod m), (\ell_b^2-1, 1), \ldots\}_{k+1}$$

New input enters the buffer:

a enters: $S_2^a = \{\ldots, (\ell_a^2+1, (i_a+1) \bmod m), (\ell_b^2-1, 1), \cdots\}_k$

b enters: $S_2^b = \ldots, (\ell_a^2, (i_a+1 \bmod m), (\ell_b^2, 1), \cdots\}_k$

j, j≠a,b enters:

$$S_2^j = \{\ldots, (\ell_a^2, (i_a+1) \bmod m), (\ell_b^2-1, 1), \cdots\}_k$$

By the induction hypothesis,

$$EC_{max}(S_1^a)_k < EC_{max}(S_2^a)_k$$

$$EC_{max}(S_1^b)_k < EC_{max}(S_2^b)_k$$

$$EC_{max}(S_1^j)_k < EC_{max}(S_1^j)_k \qquad \forall\ j \neq a, b$$

Therefore:

$$EC_{max}(S_1)_{k+1} < EC_{max}(S_2)_{k+1}$$

(4) $\ell_a^2 > \ell_b^1$, $i_a = 0$, $0 < i_b < m$

Due to the induction hypothesis, module a should be scheduled in $S_1$ and $S_2$.

For state $S_1$, schedule module a in this sub cycle,

$$\{\ldots, (\ell_a^1, 0), (\ell_b^1, i_b), \cdots\}_{k+1}$$
$$\Rightarrow \{\ldots, (\ell_a^1 - 1, 1), (\ell_b^1, i_b + 1) \bmod m), \cdots\}_{k+1}$$

New input enters the buffer:

a enters: $S_1^a$

$$= \{\ldots, (\ell_a^1, 1), (\ell_b^1, i_b + 1) \bmod m), \cdots\}k$$

b enters: $S_1^b$

$$= \{\ldots, (\ell_a^1 - 1, 1), (\ell_b^1 + 1, (i_b + 1) \bmod m, \cdots\}_k$$

j,j≠a,b enters: $S_1^j$

$$= \{\ldots, (\ell_a^1 -.1, 1), (\ell_b^1, (i_b + 1) \bmod m), \cdots\}_k$$

For state $S_2$, schedule module a in this sub-cycle,

$$\{\ldots, (\ell_a^2, 0), (\ell_b^2, i_b), \ldots\}_{k+1}$$
$$\Rightarrow \{\ldots, (\ell_a^2 - 1, 1), (\ell_b^2, (i_b + 1) \bmod m), \ldots\}_{k+1}$$

New input enters the buffer:

a enters: $S_2^a = \{\ldots, (\ell_a^2, 1), (\ell_b^2, (i_b + 1) \bmod m), \cdots\}_k$

b enters: $S_2^b$

$$= \{\ldots, (\ell_a^2 - 1, 1), (\ell_b^2 + 1, (i_b + 1) \bmod m), \cdots\}_k$$

j,j≠a,b enters: $S_2^j$

$$= \{\ldots, (\ell_a^2 - 1, 1), (\ell_b^2, (i_b + 1) \bmod m), \cdots\}_k$$

By the induction hypothesis:

$$EC_{max}(S_1^a) \leq EC_{max}(S_2^b);$$

$$EC_{max}(S_1^b) \leq EC_{max}(S_2^a);$$

$$EC_{max}(S_1^j) \leq EC_{max}(S_2^j) \qquad \forall j \neq a,b$$

Therefore:

$$EC_{max}(S_1)_{k+1} \leq EC_{max}(S_2)_{k+1}.$$

(5) $\ell_a^2 = \ell_b^1$ Both modules are not scheduled in the current sub-cycle. With the similar reasons as in III(b)(1), there exists a module z which is scheduled in the current sub-cycle. Because of the symmetry between the states of modules a and b, by the induction hypothesis II(b),

$$EC_{max}(S_1^a)_k = EC_{max}(S_2^b)_k$$

$$EC_{max}(S_1^b)_k = EC_{max}(S_2^a)_k \text{ and}$$

$$EC_{max}(S_1^j)_k = EC_{max}(S_2^j)_k \qquad j \neq a,b$$

Therefore:

$$EC_{max}(S_1)_{k+1} = EC_{max}(S_2)_{k+1}$$

(6) $\ell_a^2 = \ell_b^1$ There exists a module x such that $i_x=0$ and

$$\ell_a^2 > \ell_x > \ell_a^1 \qquad \text{if } i_a=0 \text{ or}$$

$$\ell_b^1 > \ell_x > \ell_b^2 \qquad \text{if } i_b=0.$$

For the first case,

$$S_1 = \{\ldots,(\ell_a^1, 0),(\ell_b^1,i_b), \ \ldots,(\ell_x,0), \ \cdots\}_{k+1}$$

$$S_2 = \{\ldots,(\ell_a^2, 0),(\ell_b^2,i_b),\ldots, \ (\ell_x,0),\ldots\}_{k+1}.$$

With a similar argument as in III(b)(2), suppose module x is not scheduled in both states and module a is scheduled.

Due to the symmetry between the states of module a and b, and by the induction hypothesis II(b),

$$EC_{max}(S_1 | \, a \text{ scheduled})_k = EC_{max}(S_2 | \, a \text{ scheduled})_k.$$

However, due to the induction hypothesis, II(a), scheduling x in state $S_1$ would be better than scheduling a because $\ell_x > \ell_a^1$.

$$EC_{max}(S_1 | \, x \text{ scheduled})_k < EC_{max}(S_1 | \, a \text{ scheduled}).$$
$$< EC_{max}(S_2 | \, a \text{ scheduled})_k$$

Therefore:

$$EC_{max}(S_1)_{k+1} < EC_{max}(S_2)_{k+1}$$

The other case, i.e., $\ell_b^1 > \ell_x > \ell_b^2$ and $i_b$=0 can be similarly proved.

(7) $\ell_a^2 = \ell_b^1$, $0 = i_a < i_b < m \qquad 0 = i_b < i_a < m$

The proof is very similar to III(b)(3) an d III(b)(4), except in this case, $\ell_a^2 = \ell_b^2$. Therefore, the states $S_1$ and $S_2$ are symmetric in the states of the modules a and b. By the same argument as in the proof of I(b)(3), the probability that $i_b$=0, $i_a < m$ is equal to the probability that $i_a$=0, $i_b < m$. This implies:

$$EC_{max}(S_1)_{k+1} = EC_{max}(S_2)_{k+1}$$

From the above seven cases, it is seen that in all cases,

$$EC_{max}(S_1)_{k+1} \leq EC_{max}(S_2)_{k+1}.$$

Therefore, by induction, part (b) of the theorem is proved. Because part (a) of the theorem utilizes the result of part (b) of the theorem, part (a) of the theorem is proved.

Q.E.D.

The above theorem has demonstrated that algorithm MWFMF is optimal in the sense that it minimizes the average completion time for a fixed set of random requests. Intuitively, algorithm MWFMF is better because it tries to keep all the modules as busy as possible. Suppose that some of the modules are requested more often than others. The requests to these more frequently requested modules become a bottleneck to the system whatever scheduling algorithms are used. However, a better scheduling algorithm should make use of the free cycles to schedule some requests for the less popular modules so that these requests would not accumulate after the processing of the more popular requests. This is the deficiency that occurs in other algorithms and is overcome by the MWFMF algorithm.

In addition to proving that the MWFMF algorithm has the best average case behavior, it may be necessary to show that the algorithm also possesses the best-case behavior and the best worst-case behavior. However in this case, the best-case and the worst-case behavior are identical for all algorithms. The best-case behavior occurs when all the requests are made in a sequential order, that is, 0, 1, ..., m-1, 0, 1, ..., m-1, etc. No contention would occur and the throughput of the memory is maximized, that is 1 request serviced every sub-cycle. On the other hand, the worst case behavior occurs when all the requests are directed to a single module. In this case, the bottleneck is at the module and the throughput of the memory is 1 request serviced every m sub-cycles. Algorithm MWFMF is better than other algorithms because it has a better average case behavior even though its best- and worst-case behavior are identical to that of the other algorithms.

Although the expected maximum completion time of the algorithm is minimized, it is not possible to make a similar conclusion as in Lemma 2.2 that the expected throughput of the memory is maximized because in this case, there is no relation between the expected maximum completion time and the expected throughput of the system. Furthermore, it is not useful to prove a similar theorem for the $\sum C_j$ case as in Lemma 2.3 because it is unclear that the objective of minimizing $\sum E(C_j)$ will be of any meaningful value.

Although Theorem 1 establishes that fact that the MWFMF algorithm is optimal, no throughput values are obtained analytically. In the next two sections, the throughput of the system is evaluated by using two techniques, embedded Markov Chains and simulations.

## 2.6 Simulation Technique

### 2.6.1 Simulation Results

Due to the difficulties mentioned in the last section, our evaluations are based on simulations. The simulations are run on a CDC 6400 computer. The simulation program was written in Fortran and the total time to generate all the results took over 12 hours on the CDC 6400.

Table 2.1 shows the results of simulation runs on Organization I for the memory utilization and the average waiting cycles where a waiting cycle is defined similar to Flores [FL064] as the ratio of the waiting time and the memory cycle time. Two types of request sequences are considered, one in which the requests are generated randomly, and one in which the requests are derived from the execution trace of a program. The traces used have a size of 500,000 and were obtained by running a scientific Fortran program derived from BMD applications on a CDC 7600 and they personify program characteristics of scientific applications. They have the following characteristics:

Table 2.1a. Simulation Results for Organization I with RR Scheduling Algorithm (95% confidence interval shown assuming normal distribution)

| | | Random Request Model | | Trace Driven Model | |
|---|---|---|---|---|---|
| m | b | E(Memory Utilization) | E(Waiting Cycles) | E(Memory Utilization) | E(Waiting Cycles) |
| 2 | 0 | 0.668±0.0 | 1.75±0.0 | 0.727±0.003 | 1.69±0.0 |
| | 1 | 0.801±0.017 | 2.25±0.04 | 0.882±0.003 | 2.13±0.01 |
| | 2 | 0.858±0.001 | 2.75±0.0 | 0.928±0.003 | 2.62±0.33 |
| | 3 | 0.890±0.004 | 3.25±0.02 | 0.960±0.004 | 3.08±0.52 |
| 4 | 0 | 0.401±0.004 | 1.62±0.01 | 0.472±0.026 | 1.53±0.02 |
| | 1 | 0.565±0.015 | 1.89±0.02 | 0.636±0.043 | 1.79±0.07 |
| | 2 | 0.667±0.009 | 2.12±0.02 | 0.732±0.050 | 2.03±0.14 |
| | 3 | 0.726±0.007 | 2.38±0.02 | 0.825±0.059 | 2.21±0.23 |
| 8 | 0 | 0.222±0.002 | 1.56±0.0 | 0.276±0.026 | 1.45±0.06 |
| | 1 | 0.363±0.006 | 1.69±0.01 | 0.432±0.041 | 1.58±0.07 |
| | 2 | 0.461±0.005 | 1.81±0.01 | 0.525±0.049 | 1.72±0.10 |
| | 3 | 0.534±0.006 | 1.94±0.01 | 0.610±0.060 | 1.82±0.13 |
| 12 | 0 | 0.154±0.003 | 1.54±0.0 | 0.186±0.026 | 1.45±0.05 |
| | 1 | 0.266±0.005 | 1.63±0.01 | 0.306±0.042 | 1.55±0.10 |
| | 2 | 0.354±0.005 | 1.71±0.01 | 0.408±0.058 | 1.61±0.11 |
| | 3 | 0.423±0.008 | 1.79±0.01 | 0.484±0.070 | 1.69±0.10 |
| 16 | 0 | 0.117±0.002 | 1.53±0.01 | 0.157±0.015 | 1.40±0.05 |
| | 1 | 0.209±0.003 | 1.60±0.01 | 0.254±0.024 | 1.49±0.05 |
| | 2 | 0.285±0.003 | 1.66±0.01 | 0.345±0.033 | 1.54±0.06 |
| | 3 | 0.350±0.004 | 1.71±0.01 | 0.412±0.039 | 1.61±0.09 |

Table 2.1b. Simulation Results for Organization I with FFF Scheduling Algorithm (95% confidence interval shown assuming normal distribution

| | | Random Request Model | | Trace Driven Model | |
|---|---|---|---|---|---|
| m | b | E(Memory Utilization) | E(Waiting Cycles) | E(Memory Utilization) | E(Waiting Cycles) |
| 2 | 0 | 0.501±0.0 | 2.00±0.0 | 0.571±0.002 | 1.88±0.0 |
| | 1 | 0.668±0.014 | 2.50±0.05 | 0.789±0.003 | 2.27±0.11 |
| | 2 | 0.750±0.001 | 3.00±0.0 | 0.865±0.003 | 2.73±0.34 |
| | 3 | 0.802±0.003 | 3.50±0.02 | 0.924±0.003 | 3.17±0.53 |
| 4 | 0 | 0.289±0.003 | 1.86±0.01 | 0.316±0.018 | 1.79±0.04 |
| | 1 | 0.407±0.011 | 2.23±0.04 | 0.476±0.027 | 2.05±0.12 |
| | 2 | 0.489±0.007 | 2.53±0.04 | 0.600±0.041 | 2.25±0.25 |
| | 3 | 0.544±0.008 | 2.84±0.06 | 0.678±0.048 | 2.48±0.42 |
| 8 | 0 | 0.173±0.002 | 1.72±0.01 | 0.184±0.017 | 1.68±0.06 |
| | 1 | 0.264±0.004 | 1.95±0.02 | 0.304±0.029 | 1.82±0.12 |
| | 2 | 0.330±0.005 | 2.14±0.03 | 0.379±0.037 | 1.99±0.16 |
| | 3 | 0.378±0.003 | 2.32±0.03 | 0.441±0.043 | 2.13±0.20 |
| 12 | 0 | 0.126±0.002 | 1.66±0.01 | 0.147±0.023 | 1.57±0.06 |
| | 1 | 0.201±0.003 | 1.83±0.01 | 0.235±0.033 | 1.71±0.11 |
| | 2 | 0.258±0.002 | 1.97±0.02 | 0.305±0.042 | 1.82±0.13 |
| | 3 | 0.303±0.004 | 2.10±0.03 | 0.365±0.051 | 1.91±0.17 |
| 16 | 0 | 0.100±0.001 | 1.63±0.01 | 0.106±0.010 | 1.59±0.05 |
| | 1 | 0.163±0.002 | 1.77±0.01 | 0.187±0.017 | 1.67±0.08 |
| | 2 | 0.211±0.003 | 1.89±0.01 | 0.256±0.024 | 1.73±0.10 |
| | 3 | 0.252±0.003 | 1.99±0.02 | 0.314±0.030 | 1.80±0.13 |

Table 2.1c. Simulation Results for Organization I with MWFMF Scheduling Algorithm (95% confidence interval shown assuming normal distribution

| | | Random Request Model | | Trace Driven Model | |
|---|---|---|---|---|---|
| m | b | E(Memory Utilization) | E(Waiting Cycles) | E(Memory Utilization) | E(Waiting Cycles |
| 2 | 0 | 0.667±0.008 | 1.75±0.01 | 0.727±0.01 | 1.69±0.0 |
| | 1 | 0.800±0.0 | 2.25±0.0 | 0.882±0.003 | 2.13±0.11 |
| | 2 | 0.859±0.003 | 2.75±0.03 | 0.928±0.003 | 2.62±0.33 |
| | 3 | 0.888±0.001 | 3.25±0.02 | 0.960±0.003 | 3.08±0.04 |
| 4 | 0 | 0.479±0.003 | 1.52±0.0 | 0.515±0.0 | 1.49±0.02 |
| | 1 | 0.612±0.003 | 1.82±0.01 | 0.673±0.043 | 1.74±0.08 |
| | 2 | 0.691±0.004 | 2.09±0.02 | 0.776±0.053 | 1.97±0.18 |
| | 3 | 0.740±0.004 | 2.35±0.04 | 0.831±0.059 | 2.20±0.28 |
| 8 | 0 | 0.355±0.002 | 1.35±0.01 | 0.385±0.038 | 1.33±0.06 |
| | 1 | 0.466±0.002 | 1.54±0.01 | 0.533±0.052 | 1.47±0.08 |
| | 2 | 0.544±0.004 | 1.69±0.01 | 0.612±0.058 | 1.61±0.11 |
| | 3 | 0.597±0.005 | 1.84±0.02 | 0.686±0.068 | 1.73±0.16 |
| 12 | 0 | 0.295±0.002 | 1.28±0.0 | 0.330±0.052 | 1.23±0.05 |
| | 1 | 0.399±0.003 | 1.42±0.01 | 0.472±0.066 | 1.35±0.08 |
| | 2 | 0.475±0.003 | 1.53±0.01 | 0.533±0.079 | 1.45±0.10 |
| | 3 | 0.524±0.002 | 1.64±0.01 | 0.614±0.088 | 1.54±0.12 |
| 16 | 0 | 0.259±0.001 | 1.24±0.0 | 0.300±0.028 | 1.21±0.05 |
| | 1 | 0.357±0.003 | 1.35±0.01 | 0.416±0.040 | 1.30±0.07 |
| | 2 | 0.424±0.002 | 1.44±0.01 | 0.511±0.049 | 1.37±0.08 |
| | 3 | 0.476±0.002 | 1.53±0.01 | 0.570±0.055 | 1.44±0.10 |

| | |
|---|---|
| fraction of instruction word fetches | 0.597 |
| fraction of data word fetches | 0.336 |
| fraction of data word stores | 0.067 |
| average number of accesses per inst. executed | 0.600 |
| number of instructions per instruction word | 2.787 |
| fraction of instructions that need data | 0.242 |
| fraction of instructions that are | |

|  |  |  |
|---|---|---|
| unconditional jumps |  | 0.044 |
| successful conditional jumps |  | 0.030 |
| unsuccessful conditional jumps |  | 0.015 |
| number of instructions executed between |  |  |
| conditional jumps | mean | 22.3 |
|  | st'd dev. | 10.3 |
| unconditional jumps | mean | 22.8 |
|  | st'd dev. | 24.7 |
| successful conditional jumps | mean | 33.9 |
|  | st'd dev. | 19.2 |
| all dependent events | mean | 11.4 |
| (Cond. + uncond. jumps) | st'd dev. | 10.1 |

In Table 2.2, the simulation results for Organization II are shown. Since the existence of multiple sets of buffers allows a request at $B_T$ to be blocked by a set of full buffers in a module while buffers of other modules may be empty, a column has been included in Table 2.2 to show the buffer utilization (this excludes the buffer $B_T$). The queue utilization results shown in Table 2.2 are normalized with respect to the buffer size b.

## 2.6.2 Application of Multiple Linear Regression to Obtain a Closed Form Formula

Using the results of the simulations and the assumption that the utilization is approximately 1 when $b \gg m$ (e.g., $b=100$, $m=4$), multiple linear regression is applied to fit a curve to the results [DRA66]. Based on the tail area of the partial F-value for testing the null hypothesis that a regression coefficient is zero, some of the terms in the polynomial have been eliminated. In Table 2.3, the coefficients for the regression analysis on the utilization and the waiting cycles of the two organizations under MWFMF scheduling algorith are shown. The errors in the estimation can be shown to be less than 4% in most cases except for a few cases with b=0, where the error gets to around 10%. From the polynomial equation we have obtained, we can extrapolate our results beyond b=3. The errors in extrapolating the values of utilization are small because the asymptotic value of utilization when b is large is known. However, the errors may be large when extrapolating the values of waiting cycle because its asymptotic values are not known. With Organization I, $e_B=1$, and therefore the values of waiting cycles can be derived from the values of utilization by applying Little's Formula. With Organization II, $e_B<1$, and the values of $u_{m,b}$ and $w_{m,b}$ must be known in order to estimate $e_B$. Since asymptotic values of $w_{m,b}$ and $e_B$ do not exist, the errors may be large in this case.

In Figures 2.5 to 2.10 the performance of Organizations I and II are shown. The actual simulation results are used for $b\leq 3$ while extrapolations are made for b>3. In Figure 2.5, a plot of the improvement in memory utilization with buffer size for Organization I with m = 8 is shown. It is seen that the improvement in memory utilization approaches a constant rate as the buffer size is increased. Further, the MWFMF algorithm gives the best performance. In Figure 2.6, a plot of the expected waiting cycles for different buffer sizes of Organization I is shown for m = 8. It is seen that the increase of waiting cycles is much slower than the increase of buffer sizes and the increase is almost linear. The trace driven simulation results show a higher improvement in memory utilization and a smaller number of waiting

Table 2.2a. Simulation Results for Organization II with RR Scheduling Algorithm (95% confidence interval shown assuming normal distribution)

| | | Random Request Model | | | Trace Driven Model | | |
|---|---|---|---|---|---|---|---|
| m | b | E(Memory Utilization) | E(Waiting Cycles | E(Buffer Utilization) | E(Memory Utilization) | E(Waiting Cycles | E(Buffer Utilization) |
| 2 | 0 | 0.667±0.0 | 1.75±0.0 | - | 0.727±0.003 | 1.69±0.0 | - |
| | 1 | 0.801±0.004 | 2.50±0.02 | 0.700±0.006 | 0.882±0.003 | 2.43±0.16 | 0.760±0.049 |
| | 2 | 0.857±0.002 | 3.25±0.02 | 0.715±0.003 | 0.928±0.003 | 3.18±0.44 | 0.761±0.102 |
| | 3 | 0.890±0.003 | 4.00±0.01 | 0.732±0.004 | 0.960±0.003 | 3.92±0.61 | 0.768±0.142 |
| 4 | 0 | 0.401±0.004 | 1.62±0.01 | - | 0.472±0.026 | 1.53±0.02 | - |
| | 1 | 0.629±0.004 | 2.18±0.01 | 0.492±0.005 | 0.737±0.052 | 2.09±0.17 | 0.554±0.085 |
| | 2 | 0.731±0.005 | 2.75±0.03 | 0.515±0.009 | 0.847±0.058 | 2.70±0.36 | 0.597±0.136 |
| | 3 | 0.792±0.004 | 3.33±0.02 | 0.531±0.006 | 0.903±0.064 | 3.34±0.50 | 0.621±0.158 |
| 8 | 0 | 0.222±0.002 | 1.56±0.0 | - | 0.276±0.026 | 1.45±0.06 | - |
| | 1 | 0.467±0.006 | 1.97±0.01 | 0.347±0.006 | 0.586±0.055 | 1.87±0.11 | 0.384±0.063 |
| | 2 | 0.626±0.006 | 2.41±0.02 | 0.379±0.008 | 0.793±0.078 | 2.40±0.25 | 0.494±0.113 |
| | 3 | 0.705±0.004 | 2.66±0.04 | 0.397±0.006 | 0.862±0.085 | 2.95±0.45 | 0.518±0.148 |
| 12 | 0 | 0.154±0.003 | 1.54±0.0 | - | 0.186±0.026 | 1.45±0.05 | - |
| | 1 | 0.417±0.005 | 1.88±0.01 | 0.283±0.005 | 0.599±0.083 | 1.73±0.10 | 0.354±0.070 |
| | 2 | 0.569±0.007 | 2.26±0.02 | 0.318±0.007 | 0.733±0.105 | 2.22±0.26 | 0.404±0.125 |
| | 3 | 0.661±0.005 | 2.66±0.02 | 0.338±0.006 | 0.753±0.109 | 2.63±0.47 | 0.381±0.151 |
| 16 | 0 | 0.117±0.002 | 1.53±0.01 | - | 0.157±0.015 | 1.40±0.05 | - |
| | 1 | 0.379±0.005 | 1.82±0.01 | 0.249±0.004 | 0.502±0.048 | 1.73±0.09 | 0.303±0.049 |
| | 2 | 0.534±0.008 | 2.18±0.02 | 0.283±0.02 | 0.692±0.066 | 2.05±0.17 | 0.333±0.075 |
| | 3 | 0.626±0.004 | 2.54±0.08 | 0.300±0.005 | 0.745±0.072 | 2.23±0.32 | 0.284±0.094 |

Table 2.2b. Simulation Results for Organization II with FFF Scheduling Algorithm (95% confidence interval shown assuming normal distribution)

| | | Random Request Model | | | Trace Driven Model | | |
|---|---|---|---|---|---|---|---|
| m | b | E(Memory Utilization) | E(Waiting Cycles | E(Buffer Utilization) | E(Memory Utilization) | E(Waiting Cycles | E(Buffer Utilization) |
| 2 | 0 | 0.501±0.0 | 2.00±0.0 | - | 0.571±0.002 | 1.88±0.0 | - |
| | 1 | 0.670±0.004 | 2.74±0.0 | 0.669±0.004 | 0.789±0.003 | 2.56±0.16 | 0.732±0.045 |
| | 2 | 0.751±0.003 | 3.50±0.02 | 0.688±0.010 | 0.865±0.003 | 3.30±0.48 | 0.743±0.097 |
| | 3 | 0.789±0.002 | 4.27±0.01 | 0.699±0.004 | 0.924±0.003 | 4.00±0.62 | 0.758±0.138 |
| 4 | 0 | 0.289±0.003 | 1.86±0.01 | - | 0.316±0.018 | 1.79±0.04 | - |
| | 1 | 0.458±0.005 | 2.54±0.01 | 0.454±0.007 | 0.557±0.042 | 2.38±0.22 | 0.519±0.095 |
| | 2 | 0.558±0.004 | 3.19±0.03 | 0.483±0.009 | 0.702±0.048 | 2.95±0.48 | 0.558±0.149 |
| | 3 | 0.628±0.003 | 3.80±0.04 | 0.503±0.007 | 0.801±0.057 | 3.54±0.72 | 0.596±0.184 |
| 8 | 0 | 0.173±0.002 | 1.72±0.01 | - | 0.184±0.017 | 1.68±0.06 | - |
| | 1 | 0.329±0.005 | 2.32±0.03 | 0.311±0.010 | 0.406±0.040 | 2.20±0.19 | 0.360±0.069 |
| | 2 | 0.436±0.002 | 2.88±0.02 | 0.348±0.004 | 0.604±0.060 | 2.73±0.36 | 0.461±0.121 |
| | 3 | 0.498±0.004 | 3.44±0.07 | 0.363±0.011 | 0.671±0.067 | 3.36±0.65 | 0.492±0.162 |
| 12 | 0 | 0.126±0.002 | 1.66±0.01 | - | 0.147±0.023 | 1.57±0.06 | - |
| | 1 | 0.278±0.004 | 2.20±0.03 | 0.250±0.007 | 0.396±0.055 | 2.03±0.18 | 0.325±0.080 |
| | 2 | 0.383±0.005 | 2.71±0.05 | 0.285±0.010 | 0.510±0.073 | 2.64±0.43 | 0.376±0.137 |
| | 3 | 0.457±0.004 | 3.22±0.05 | 0.310±0.009 | 0.529±0.085 | 3.19±0.55 | 0.358±0.752 |
| 16 | 0 | 0.100±0.001 | 1.63±0.01 | - | 0.106±0.010 | 1.59±0.05 | - |
| | 1 | 0.252±0.003 | 2.11±0.02 | 0.217±0.005 | 0.345±0.032 | 1.96±0.14 | 0.267±0.050 |
| | 2 | 0.349±0.003 | 2.61±0.03 | 0.250±0.005 | 0.478±0.046 | 2.33±0.30 | 0.286±0.082 |
| | 3 | 0.424±0.004 | 3.09±0.05 | 0.141±0.008 | 0.527±0.050 | 2.62±0.46 | 0.262±0.092 |

Table 2.2c. Simulation Results for Organization II with MWFMF Scheduling Algorithm (95% confidence interval shown assuming normal distribution)

| | | Random Request Model | | | Trace Driven Model | | |
|---|---|---|---|---|---|---|---|
| m | b | E(Memory Utilization) | E(Waiting Cycles | E(Buffer Utilization) | E(Memory Utilization) | E(Waiting Cycles | E(Buffer Utilization) |
| 2 | 0 | 0.667±0.008 | 1.75±0.01 | - | 0.727±0.003 | 1.69±0.0 | - |
| | 1 | 0.799±0.003 | 2.50±0.01 | 0.700±0.005 | 0.882±0.003 | 2.43±0.16 | 0.760±0.049 |
| | 2 | 0.856±0.005 | 3.25±0.01 | 0.714±0.007 | 0.928±0.003 | 3.18±0.44 | 0.761±0.102 |
| | 3 | 0.890±0.020 | 4.00±0.02 | 0.724±0.006 | 0.960±0.003 | 3.92±0.61 | 0.768±0.142 |
| 4 | 0 | 0.419±0.003 | 1.52±0.0 | - | 0.515±0.029 | 1.49±0.02 | - |
| | 1 | 0.648±0.001 | 2.13±0.01 | 0.482±0.002 | 0.738±0.052 | 2.07±0.18 | 0.539±0.090 |
| | 2 | 0.743±0.003 | 2.72±0.01 | 0.515±0.005 | 0.838±0.059 | 2.70±0.36 | 0.588±0.135 |
| | 3 | 0.795±0.002 | 3.31±0.02 | 0.528±0.003 | 0.902±0.032 | 3.35±0.53 | 0.625±0.157 |
| 8 | 0 | 0.355±0.002 | 1.35±0.01 | - | 0.385±0.038 | 1.33±0.06 | - |
| | 1 | 0.534±0.005 | 1.85±0.01 | 0.328±0.007 | 0.624±0.062 | 1.84±0.13 | 0.398±0.077 |
| | 2 | 0.651±0.003 | 2.33±0.01 | 0.371±0.003 | 0.799±0.079 | 2.40±0.26 | 0.498±0.118 |
| | 3 | 0.717±0.003 | 2.81±0.02 | 0.391±0.004 | 0.849±0.083 | 2.95±0.44 | 0.510±0.144 |
| 12 | 0 | 0.295±0.002 | 1.28±0.0 | - | 0.365±0.052 | 1.23±0.05 | - |
| | 1 | 0.472±0.005 | 1.72±0.01 | 0.256±0.005 | 0.624±0.089 | 1.68±0.14 | 0.343±0.095 |
| | 2 | 0.602±0.004 | 2.16±0.01 | 0.308±0.004 | 0.735±0.106 | 2.19±0.29 | 0.395±0.135 |
| | 3 | 0.683±0.006 | 2.58±0.03 | 0.332±0.009 | 0.756±0.109 | 2.61±0.49 | 0.377±0.154 |
| 16 | 0 | 0.259±0.001 | 1.24±0.0 | - | 0.300±0.028 | 1.21±0.05 | - |
| | 1 | 0.439±0.006 | 1.64±0.01 | 0.217±0.007 | 0.565±0.054 | 1.62±0.10 | 0.289±0.061 |
| | 2 | 0.564±0.007 | 2.05±0.02 | 0.264±0.007 | 0.692±0.066 | 1.97±0.21 | 0.306±0.083 |
| | 3 | 0.647±0.003 | 2.44±0.02 | 0.290±0.004 | 0.745±0.072 | 2.17±0.34 | 0.271±0.096 |

Table 2.3 - Coefficients of 3rd Order Polynomial Regression of Organization I and II under MWFMF Scheduling Algorithm (RRM - Random Request Model; TDM - Trace Driven Model) *Note: All other coefficients are set to zero*

Utilization

| Model | $m^2$ | m | 1/m | $b^{1/3}$ | $b^{1/2}$ | $b^{1/4}$ | $\frac{b^{1/3}}{m}$ | $\frac{b^{1/2}}{m}$ | const. |
|---|---|---|---|---|---|---|---|---|---|
| RRM-I | 0.00050 | -0.02011 | 0.58124 | 1.80176 | -0.32495 | -1.37185 | 0.27655 | -0.21970 | 0.41273 |
| TDM-I | 0.00065 | -0.02312 | 0.62605 | 2.29106 | -0.45177 | -1.69628 | 0.18115 | -0.18268 | 0.45447 |
| RRM-II | -0.00009 | -0.00283 | 0.79465 | 3.04862 | -0.64641 | -2.17849 | -0.22013 | 0.00866 | 0.26880 |
| TDM-II | -0.00012 | -0.00301 | 0.80863 | 2.72327 | -0.61966 | -1.80155 | -0.44904 | 0.11118 | 0.33023 |

Waiting Cycles

| Model | $m^3$ | $m^2$ | $m^2b$ | m | b | mb | const. |
|---|---|---|---|---|---|---|---|
| RRM-I | -0.00109 | 0.03312 | 0.00314 | -0.31779 | 0.61021 | -0.08138 | 2.30046 |
| TDM-I | -0.00100 | 0.03038 | 0.00308 | -0.29282 | 0.56690 | -0.07881 | 2.19725 |
| RRM-II | -0.00082 | 0.02570 | 0.00219 | -0.26252 | 0.84230 | -0.06200 | 2.19883 |
| TDM-II | -0.00075 | 0/02432 | 0.00016 | -0.25363 | 0.77708 | 0.03024 | 2.12700 |

cycles than the random request model. This is because there is a higher correlation between consecutive requests and the requests are likely to be made in a consecutive order. As a result, there is less contention in the system. The curves showing the estimated results due to dependencies are discussed in the following sections. The above observations are also true for other values of m. Further, the MWFMF algorithm has the minimum amount of waiting time among the three algorithms studied. In Figures 2.7 and 2.8, the decrease in memory utilization and waiting cycles for increasing degrees of interleaving of Organiation I with a MWFMF algorithm are plotted. The rate of decrease in memory utilization is more pronounced and the utilization is higher when the degree of interleaving is small. Also, the effects on waiting cycles due to buffer size is very small when the ratio of buffer size to degrees of interleaving is small. Other scheduling algorithms also possess the same properties. The effects on the memory utilization and the waiting cycles for various buffer sizes of Mode II are similar to those of Organization I. In Figure 2.9, the effects on buffer utilization are shown for various buffer sizes of Organization II. It is seen that the buffers are less utilized as the size is increased. This also accounts for the diminishing increase in memory utilization as the buffer size is increased. The difference in buffer utilization among the three scheduling algorithms is very small. However, extrapolations for values of b beyond 3 are not accurate for Organization II for reasons noted before. In Figure 2.10. a plot of buffer utilization versus different degrees of interleaving is shown. The buffer utilization drops as the number of modules is increased. However, it is seen in both Figures 2.9 and 2.10 that the buffer utilization is not sensitive to buffer size changes. The decrease in buffer utilization is due to the fact that there is a higher probability that $B_T$ is blocked when the number of modules is increased.

## 2.7 Effects of Separating the Instruction and the Data Area

The previous results have been obtained from simulations using a merged instruction and data area. Since an instruction access results in some data accesses, it is desirable to place the data accessed in modules not conflicting with the next inst.uction accessed. This motivates us to investigate the separation of instruction and data area into different modules in the main memory. Sastry et. al. [SAS75] and

Nutt [NUT77] have made some pioneering studies on the separation of instruction and data areas, but they have assumed a non-pipelined multiprocessor system. We study the effects with respect to a pipelined processor here. In this section, an organization with separate instruction and data modules is compared against an organization with merged instruction and data modules using the traces available. Consecutive instruction words are put in consecutive instruction memories and consecutive data locations are put in consecutive data memories.

The characteristics of the traces reveal that 60% of the accesses are instructions and the rest are data accesses; therefore the modules should be divided according to this ratio approximately. Since it is desirable to have the number of instruction modules and the number of data modules an integral power of 2 for ease of address decoding, the modules are divided into a 4-2 partition so that four of the modules are instruction modules and two are data modules. It is not possible to designate exactly 60% of the modules as instruction modules and to satisfy the requirement that the number be an integral power of 2. Since there are 6 modules in the 4-2 partition, it is necessary to compare the performance of the 4-2 partition against a hypothetical 6-way interleaved system with merged instruction and data modules. The results are shown in Tables 2.4 and 2.5 It is seen that the differences between the two alternatives are minimal. In fact in some cases, the merged model seems to perform a little better. This is due to the unqeal utilization and waiting cycles of the modules in the separated case. From the simulation results on the utilization of the individual modules (not shown), the instruction modules are found to be under-utilizaed while the data modules are found to be over-utilized. One way to improve the performance

Table 2.4 Comparison between Merged and Separated Instruction-Data Areas for Organization I -- Trace Driven Simulation

| | | | RR | | FFF | | MWFMF | |
|---|---|---|---|---|---|---|---|---|
| | m | b | Memory util. | Waiting cycles | Memory util. | Waiting cycles | Memory util. | Waiting cycles |
| Merged | 6 | 0 | 0.338 | 1.49 | 0.243 | 1.69 | 0.459 | 1.36 |
| Inst.-Data | | 1 | 0.501 | 1.65 | 0.403 | 1.83 | 0.624 | 1.53 |
| Areas | | 2 | 0.657 | 1.76 | 0.479 | 2.04 | 0.695 | 1.72 |
| (m=6) | | 3 | 0.726 | 1.92 | 0.543 | 2.23 | 0.752 | 1.89 |
| Separate | 6 | 0 | 0.336 | 1.50 | 0.270 | 1.62 | 0.486 | 1.34 |
| Inst.-Data | | 1 | 0.517 | 1.64 | 0.394 | 1.85 | 0.619 | 1.54 |
| Areas | | 2 | 0.618 | 1.81 | 0.484 | 2.03 | 0.692 | 1.72 |
| (4-2 ways) | | 3 | 0.696 | 1.96 | 0.540 | 2.24 | 0.730 | 1.91 |

Table 2.5 Comparison between Merged and Separated Instruction-Data Areas for Organization II - Trace Driven Simulations

| | | | RR | | | FFF | | | MWFMF | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | m | b | Mem. Util. | Wait. Cycle | Buf. Util. | Mem. Util. | Wait. Cycle | Buf. Util. | Mem. Util. | Wait. Cycle | Buf. Util. |
| Merged | 6 | 0 | 0.34 | 1.49 | - | 0.24 | 1.69 | - | 0.46 | 1.36 | - |
| Inst.-Data | | 1 | 0.69 | 1.95 | 0.49 | 0.50 | 2.23 | 0.44 | 0.69 | 1.94 | 0.48 |
| Areas | | 2 | 0.80 | 2.48 | 0.50 | 0.61 | 2.83 | 0.47 | 0.79 | 2.49 | 0.50 |
| (m=6) | | 3 | 0.81 | 2.88 | 0.45 | 0.63 | 3.27 | 0.42 | 0.81 | 2.87 | 0.45 |
| Sep. | 6 | 0 | 0.34 | 1.50 | - | 0.27 | 1.62 | - | 0.49 | 1.34 | - |
| Inst.-Data | | 1 | 0.65 | 1.98 | 0.47 | 0.49 | 2.18 | 0.40 | 0.67 | 1.90 | 0.43 |
| Areas | | 2 | 0.75 | 2.42 | 0.45 | 0.57 | 2.74 | 0.41 | 0.76 | 2.41 | 0.45 |
| (4-2 ways) | | 3 | 0.77 | 2.96 | 0.45 | 0.59 | 3.29 | 0.40 | 0.77 | 2.95 | 0.45 |

## 2.8 Degradation in Performance Due to Dependencies

In the previous sections, we have simulated the organizations under the assumption that there is a high request rate from the pipe so that any empty buffers can be replenished until they are full or a blockage occurs. However, this assumption is not totally valid in a pipelined uni-processor. As mentioned earlier, there are three sources of interference which result in emptying the pipe and reloading a new instruction stream. In the process of emptying the pipe, new memory requests are not generated and the memory becomes idle after all the pending requests are serviced. The utilization of the memory is therefore lower than our simulated results. One solution is to simulate a pipe together with the memory. However, different computers handle dependencies differently, and the simulation of a particular machine is too limited in scope. We therefore choose to estimate the resulting utilization with a general model.

### 2.8.1 The Model Used to Estimate the Performance Due to Dependencies

Without loss of generality, all dependencies can be represented as a successful (the jump is taken) or an unsuccessful conditional jump. In a conditional jump instruction, the condition code is set earlier by an instruction which may still be in the pipe. Until that instruction finishes and sets the condition code, the jump instruction cannot proceed. It is assumed that the pipe prefetches but does not decode the target instruction. If it is an unsuccessful jump, the pipe can proceed after the condition code has been set. If it is a successful jump, the pipe has to wait until both the condition code is set and the target instruction is fetched from the memory. An unconditional jump can be modelled as a succssful conditional jump in which the condition code is available immediately. A register interlock is the same as an unsuccessful conditional jump instruction and an interrupt is the same as a successful conditional jump in which the entire pipe has to be emptied.

The model used in the estimation is shown in Figure 2.11. A linear pipe is considered. The instruction prefetch unit has to fetch instructions ahead of the instruction decode unit so that the decode unit never has to wait for instruction fetches. Let

L = number of stages of the pipe;

T = time needed to pass through one a stage of the pipe;

f = the number of instruction words prefetched.

The memory is assumed to be a single server with a constant service time of rate $u_{m,b}$, and a finite buffer space of length $M$-$u_{m,b}$*m (Eq. 2.2, 2.3). The service discipline in the buffers is FIFO and the waiting time for a request is W (Eq. 2.4). Since we are interested in getting an expected value of the performance, the model is a sufficient approximation of the actual model. It is also assumed that the occurrences of successive dependent requests are separated far enough and have no effect on each other. By "far enough," it is meant that after a dependency is resolved, sufficient time elapses so that all the buffers are filled up before the occurrence of another dependent request. The maximum time needed is $u_{m,b}$*$M$ (Figure 2.14). This assumption is necessary because the effect due to each dependent request can be found separately and the overall effect due to all the dependent requests is the sum of the individual effects. From the statistics of the traces which are shown in Figures 2.12 and Figure 2.13, it is found that successive dependent requests are separated by an average of twelve instructions. Successive dependent requests may therefore have effects on each other and our analysis slightly under-estimates the actual performance.

### 2.8.2 Computation of Degradation in Performance

The effect of dependencies is measured in terms of an idle period. An idle period of the memory is defined to be a time interval during which requests to the memory stop. The idle period is measured in terms of the number of memory sub-cycles. At the beginning of an idle period, the number of requests drops gradually to zero (Figure 2.14). The resulting utilization of the module is lower as is evident from a similar model with a smaller buffer size. When the pipe starts requesting again, the number of requests in the buffers gradually builds up to the maximum amount. The idle period is defined in this way because it represents an average length of the time during which the buffers are not fully utilized. Let

$d$ = distance in terms of the number of pipe segments between the instruction setting the condition code and the conditional jump instruction at the decode segment;

$r$ = average number of requests generated per instruction executed;

$i$ = number of instructions per instruction word;

$x^{s}_{CJ}$ = fraction of instructions executed that are successful conditional jumps;

$x^{u}_{CJ}$ = fraction of instructions executed that are unsuccessful conditional jumps.

In the trace driven simulation results in Section 2.7.1, the instructions and the corresponding operands are assumed to be accessed one after the other. In the current model, the instructions are fetched much earlier than the corresponding operands. We have ignored these effects on the memory performance because there is very little correlation between the instruction address and its corresponding operand address (except in some cases; e.g., an architecture which implements the immediate mode, but the frequency of executing these instructions is small).

Since it is desired to find the maximum performance of the memory, the pipe must be designed in a way such that it is fast enough and long enough so that it is always able to fill up all the empty buffers in the memory within a memory sub-cycle. This design follows from our high request rate assumption. In this model, the pipe is essentially executing at the speed of the memory, that is, at the rate $u_{m,b}/r$. The assumptions made are:

(1) There are a large amount of return buffers in the pipe for serviced requests. This assumption is necessary so that serviced requests can always be returned to the CPU without delaying the initiation of requests in the memory.

(2) Each segment in the pipe is very fast. This means that T is so small that if sufficient instructions are available to the decode unit, the pipe can generate enough requests to fill up all the memory buffers in one memory sub-cycle. This means:

$$T^{*}M \leq \frac{\mathrm{T}}{u_{m,b}}$$

That is

$$T = \frac{\mathrm{T}}{u_{m,b}{}^{*}M} \qquad (2.5)$$

(3) Since it takes a time W ($= w_{m,b}{}^{*}m$) to fetch an instruction, the pipe would have executed i*f instructions in this time interval at a rate of $\frac{u_{m,b}}{\mathrm{T}}$ if no dependency occurs. Therefore

$$\frac{i^* f}{u_{m,b}/T} > w_{m,b}{}^* m$$

we set

$$f = \left| \frac{w_{m,b}{}^* u_{m,b}{}^* m}{i^* r} \right| \qquad (2.6)$$

where $|y|$ is the smallest integer larger than $y$. The value of f is chosen to be the smallest possible because when a conditional branch is encountered, one of the two paths is not traversed and therefore the instruction fetches for that path are wasted. The value of f is kept small in order to reduce the effects due to this waste.

(4) After an operand request is generated, the operand will be serviced after an average time W. In the meantime, the corresponding instruction passes through L-2 stages of the pipe in order to get to the execution unit. The time for this instruction to pass through the pipe must be longer than the waiting time for its operand so that the pipe is not blocked by this instruction waiting for its operand. We have

$$\frac{T^*(L-2)}{u_{m,b}} \geqq w_{m,b}{}^* m$$

We set

$$L = \left| \frac{w_{m,b}{}^* u_{m,b}{}^* m}{T} \right| + 2 \qquad (2.7)$$

The value of L set in Eq. 2.7 is the minimum pipe length required for a maximum memory performance. For a longer pipe, the memory performance is lower because it takes a longer time for a dependent request to pass through the pipe. For a shorter pipe, the pipe is not able to generate requests fast enough because the last stage of the pipe is frequently blocked by unfinished operand requests. The value of L chosen is therefore a compromise between these two effects. These additional constraints can assure that the maximum performance of the memory is achieved.

When a conditional jump instruction is encountered and the condition code is set at a distance d stages away, the execution of the conditional jump is stopped until the instruction setting the condition code passes through L-d segments at a rate of $u_{m,b}/T$, if the conditional jump is unsuccessful. However, if it is successful, then the pipe is blocked until both the condition code is set and the target instruction has been fetched from the memory. If $t_0^{s/f}$ is set to be the time interval between the recognition of a successful/unsuccessful conditional jump and the time when the pipe can start execution again, then

$$t_e^{s} = \max\{(L - d)*\frac{T}{u_{m,b}} , w_{m,b}*m\} \tag{2.8a}$$

$$t_e^{f} = (L - d)*\frac{r}{m} \tag{2.8b}$$

After the jump has been determined, it takes a small amount of time T/r to generate the operand request. It is not assumed that the decoding is done beforehand as in some machines. Let $t_T^{s/f}$ be the time interval from the recognition of a successful/unsuccessful conditional jump to the time when the pipe starts making requests. Then

$$t_T^{s/f} = t_e^{s/f} + \frac{T}{T} \tag{2.9}$$

After a dependent instruction has been encountered, there are still f instruction prefetch requests in the pipe. The idle period begins after these requests have been made to the memory. Let $t_b$ be the time to make these remaining requests.

$$t_b = f * \frac{1}{u_{m,b}} \tag{2.10}$$

The length of the idle period ($ip^{s/f}$) is therefore the difference between $t_T$ and $t_b$.

$$ip^{s/f} = \max\{0, t_T^{s/f} - t_b\} \tag{2.11}$$

The above analysis is true for a particular value of d. Let D be the random variable denoting the distance, and D has the following distribution

$$PT(D=d) = \begin{cases} p_d & d = 1,2,\ldots,L \\ 0 & otherwise \end{cases} \tag{2.12}$$

This distribution is shown for the traces in Figure 2.13.

As a result of the degradation in memory utilization, there is a degradation in the buffer utilization. During and idle period, requests to the memory stop. At the end of the idle period, requests to the memory begin again. In Figure 2.14, the decrease and the increase in the number of requests in the buffer are shown. Since M may not be an integer in our model (effective buffer length in Organization II), a linear approximation is used in the original function. In terms of the idle period, the time interval during which the buffers are not full is $y = ip^{s/f} + (M - M_e)*u_{m,b}$ where $M_e$ is the effective number in the system. In fact, the shaded blocks in Figure 2.14 can be rearranged so that the effective buffer utilization can be calculated. $M_e$, during an idle period in the two cases, is

$$M_e = \begin{cases} M - \dfrac{ip^{s/\delta}}{u_{m,b}} & \text{if } ip^{s/\delta} < M^* u_{m,b} \\ 0 & \text{if } ip^{s/\delta} \geq M^* u_{m,b} \end{cases} \qquad (2.13)$$

Using the statistics from the trace program, the results of the estimated utilization are plotted together with the simulation results in Figures 2.5 and 2.7. The degradation is quite significant and drops to about 50% of the original value in some cases. As seen in Figure 2.5, the module utilization levels off much more rapidly with increasing buffer size than the original results with no dependencies. The curves plotted are not smooth because of the integrality requirement in the pipe length and the number of prefetched instructions. It is further seen that increasing buffer sizes do not improve the performance due to the effects of dependency. The difference in memory utilization for b = 3 and b = 10 is very small as seen in Figure 2.7. The estimations for waiting cycles are not plotted in Figures 2.6 and 2.8 because they coincide almost exactly with the simulation results. In Figure 2.15, the buffer utilization for Organization I with an MWFMF algorithm is plotted. It is seen that the buffer utilization is almost constant for large values of m. It is also interesting to note that the buffer utilization is lower for larger values of b. The explanation for this is because for a large value of b, the waiting time in the memory is longer and the memory utilization is higher. This implies that a longer pipe must be used (Eq. 2.7). A longer pipe means that it takes longer to resolve a dependent request and this causes degradation in the buffer utilization.

The above estimations only give an average value for the performance. In fact, if the memory can be utilized in some other way (e.g., for peripheral processing) when a dependency occurs, the degradation may not be so significant. The above analysis also reveals the fact that when the occurrences of dependent requests are frequent, it is not beneficial to use a pipelined computer in a batch mode. High degree of program interleaving using multiprogramming would help in reducing the degradation due to dependencies.

## 2.9 Some Final Remarks about the Design of Interleaved Memories

We have presented in this section two organizations of an interleaved memory system which utilizes a finite buffer space for the storage of requests. We have designed a scheduling algorithm which allows a finite set of requests to be processed in the minimum expected time. However, the performance of our system is obviously less than the performance of systems with an infinite saturated request queue which is an unrealistic assumption. In Figure 2.7, we have shown the performance of Hellerman's model [HEL67] together with our simulation results. Although Hellerman's model is a simple model and allows no queueing of requests, it is useful as a lower bound for the performance of other systems. It is seen that with a random request queue, Hellerman's model is better than our Organization I with $b = 0$, but is worse for $b > 0$. Note that the performance curves all have the same shape. Since Organization II degenerates into Organization I for $b = 0$, it is worse than Hellerman's model for $b = 0$, but better for $b > 0$. The comparison with other models in the current literature is not meaningful because they differ significantly.

We can improve our model slightly by considering the following. The rationale behind the constraint that only one module may be initiated in any sub-cycle is because the return bus can return at most one piece of datum in any sub-cycle. But since reads generate return data while writes do not, we can initiate two or more modules in a sub-cycle provided that exactly one of the requests is a read. The improvement in utilization due to this is only about 2%. The improvement is not significant because the fraction of writes in our trace is less than 7% of all the accesses and its applicability is also limited by memory interference.

The questions that still remain to be resolved are how can one select between Organization I and Organization II and how does one choose the parameters of the system in order to satisfy all the requirements. In the hardware requirements, Organization I needs associative search capabilities in the buffers while Organization II does not. However, the availability of fast associative memory can help in this regard. The performance of Organization II predicted may be worse because it may require the transfer of more than one request into the memory system during a memory sub-cycle and it sometimes is not possible in a pipelined system. Organization II gives a slightly worse performance than Organization I when a maximum of one request is allowed to be generated in each sub-cycle and the effective buffer sizes in both organizations are identical. Tradeoff in cost and performance must be made in the selection of the organization. In order to answer the second question we have raised, we need to design a cost model of the system. The cost of individual components is highly technology dependent and will not be discussed here. However, the designer can find a configuration with the minimum cost based on the band width and the response time requirements. Assuming that the bus width is determined and fixed, he can use the average utilization (a function of the degrees of interleaving) as an alternate measure of bandwidth. The response time can also be normalized with respect to the speed of the memory to give the waiting cycle. In the above calculations, the effects of dependency are not considered; otherwise Equations 2.6 and 2.9 can be used to find the values of utilization and waiting cycle dependency. Using Little's Formula, the average number of requests in the memory, or the average number in the request buffers can be obtained. The designer can then substitute the values for the average utilization and the buffer size into the formula obtained by regression (Table 2.3) to get a polynomial equation as a function of the degrees of interleaving and the memory speed. By evaluating the speed for different possible degrees of interleaving, the cost of the memory can be estimated. The final configuration selected will be the one with the minimum cost.

## 3. Data Compression

With the increase in the amount of information processing, it is important to keep the utilization of the memory high. The information content of data stored in large alphanumeric data bases is usually low. Further, as the processing becomes distributed, the communication overhead of transferring data from one location to another is usually substantial. In order to keep the utilization of the storage subsystem high, and to keep the amount of data transferred over communication links low, data compression is a natural solution to the problem. However, the use of compression codes which remove the redundancy of data seems to be in direct conflict with the use of redundant coding, e.g., parity check codes, which increase the reliability. What is needed then is an exploration of efficient error limiting codes which can be applied to compressed data and an analysis of the error rate of various compression schemes.

## 3.1 Desirable Properties of Compression Codes

In designing a compression code, it should possess to some degree each of the following properties:

1) The technique should be reversible; i.e., the original data should be fully recoverable from the compressed form. This property can be relaxed in certain situations when the data is repeated elsewhere; e.g., the keys in a directory structure are usually repeated across levels.
2) The coding scheme should cause a measurable reduction in the size of the stored data. In comparing compression codes, a standard measure called percent compression is generally used.

$$\text{percent compression} = \frac{[\text{size of input data}] - [\text{size of output data}]}{[\text{size of input data}]} \times 100\%$$

3) The technique should be reasonably efficient to implement.
4) The technique should be general enough to be equally applicable to all alphnumeric data files.

Two other properties which are often desireable in compression schemes are:

5) The prefix property; i.e., no code is the prefix of another code. This assures that the decoder never has to backup on any portion of the text.
6) Lexicographic ordering property; i.e., if the input data is in a sorted order, than after encoding, the output data is still in sorted order. This property is useful for indexes.

Existing compression techniques, which posses part of all of the above properties, can be classified into the following board categories: (1) run length encoding; (2) differencing; (3) statistical encoding; (4) value set schemes.

Run Length Encoding - In a data base, there are frequent occurrences in which the data occur in a continuous sequence of identical characters; e.g., sequence of zeroes. This sequence can be replaced by the character followed by a count. Run length encoding is a technique by which a string of continuous characters or a "clump" are replaced by a repeat flag for the character followed by the size of the "clump" or run length. In practice, however, since very long clumps are highly improbable, one can limit the run length encoded and combine the flag and length in a single byte. This is the technique used in WYLBUR [FAJ73]. Run length encoding of a single haracter type is potentially the most successful, with diminishing returns for more characters. Huang has discovered an upper bound for the entropy of run length encoding [HUA74].

Differencing - Differencing refers to techniques which compare a current record to a pattern record and retain only the differences between them. It is particularly successful with large files of records with fixed alphanumeric fields where most corresponding fields are the same or are blanks and zeroes. This is the approach normally used for sequential files, where the pattern record is taken by the previous record in the file. When differencing is applied to direct access files, however, the first record of each block is left uncompressed and used as a pattern for the remaining records in the block. The unit of information on which differencing is performed can be the bit, the byte, the field or some logical data in the record. Byte-level differencing is the most common case since byte access is convenient and cheap. In field-level differencing, bit maps are often used to indicate the presence or absence of a field when identical to the previous. Two examples of the use of differencing in relational data base systems are Titman's experimental system and the Peterless Relational Test Vehicle [TOD76].

Statistical Encoding - Statistical encoding is a transformation of an input alphabet so that it is assigned a code bit string whose length is inversely proportional to the frequency of its occurrences in the text. Since different characters occur with different frequencies, a statistical encoding scheme will usually compress the text. Huffman coding scheme is an optimum, elegant and simple algorithm to assign variable length bit codes with the prefix property to characters, given their frequencies of occurrence in a text. There are other techniques such as the Hu-Tucker Algorithm, which has both the prefix and the lexicographic ordering property. The major drawbacks of statistical encoding are that it does not exploit the natural radix of the computer (e.g., byte, word, etc.), and it does not take into account some special characteristics of the data; e.g., strings of repeating characters, and the distinction between numeric and character data. A solution to this is the use of fixed length encoding which manipulates data in units of byte [SCH71]. Further, the fact that the size of each character is variable also causes problems when the data are modified and the reliability of the data is difficult to assure because the character stream would not be recognizable once a bit is destroyed.

Value Set Schemes - A value set scheme in a data base system is a coding scheme in which repeated storage of data elements in their full character representation is avoided. Instead, each data element is stored once in the system and all subsequent occurrences of the same data element are referred to the first stored occurrence. An example of this technique is shown in the MacAIMs Data Management (MADAM) System in which a reference number is assigned to a new entering data element and all subsequent operations on the data element use the reference number. However, the fact that reference numbers are unique only within a relation could lead to problems in the reliability of the data management system and the integrity of the data. The MADAM System also uses a binary tree scheme for maintaining reference numbers which is inefficient for insertion and costly in storage space for large sets of data. There are other schemes which represent a better tradeoff between storage efficiency and processing efficiency [KNU73].

The decision of which code to use is highly dependent on the applications. For example, in a data base where the order of data is not important, the lexicographic ordering property is not important. The required properties of the applications must therefore be identified by the designers before the code is selected.

## 3.2 Future Directions of Research

While there are many reported results on data compression, the future directions of research are seen to be concentrated in the following areas:

Identify and characterize data redundancy - In a data base, there are many levels of data. For example, there are the file level, the record level, the field level and the byte level. The type of data redundancy at each level must be identified. This would aid in selecting data compression schemes best suited to the particular type of redundancy. Further, it leads us to the possibility of multi-level compression schemes, wherein data is compressed through a set of cascaded stages. Each level of the data is possibly compressed using a different technique. The compression code must be selected so that it minimizes the effects on other levels of the data.

Develop a comparison model for various compression schemes - The comparison model must be able to measure the amount of storage reduction and the computation cost for encoding and decoding. A simple measure is the percent compression defined earlier. The computation cost can be broken down into the CPU cost, the memory usage cost and the input/output cost. In order to calculate the storage reduction for a given compression scheme, the number of encodable units of tokens in a reocrd or file must be predicted. This can be obtained from an assumed input distribution such as uniform distribution, normal distribution or Zipf's distribution at the given level of data.

Study adaptive Huffman coding techniques which respond to update activity - As the data base gets updated, the initial Huffman code assignment based on the a priori character frequency distribution may no longer be optimal. A threshold for the expected compression ratio has to be determined which can dynamically reassign the variable length codes for the new frequency distribution. Further, the threshold selected should not cause excessive re-coding. The problem of updates which change the size of the data, and the reliability problems should also be studied.

Investigate the feasibility of implementing, in a microprocessor, a simple self-measuring self-adjusting encoder/decoder - Experience has shown that the current implementation of data base systems are I/O bound within a node and communication bound on the DCS. A microprocessor encoder/decoder, by performing compression and decompression, would cause communications to be done more efficiently, at the same time distributing or relieving this function from the processor sub-systems. Such a device would perform the following functions: (a) encode and decode data; (b) measure and adjust the code assignments; (c) detect errors and automatically re-initiate the operation; and (d) control concurrent accesses. The advantage of this design is that it would make data compression transparent to the rest of the system.

In conclusion, the use of data compression allows data to be stored more efficiently and data communication to be done with shorter messages. However, many issues relating to the feasibility, the design of coding techniques, the reliability of the resultant codes, the implementation issues,etc., must be solved. It is contended that such solutions do not exist now and future study is necessary.

## 4. Summary

In this report, we have discussed the design of interleaved memory system. We present two different implementation alternatives of interleaved memories (Organization I and Organization II). The two organizations differ in the configurations of the request buffers. In Organization I, a single set of request buffers is assumed to be shared by all the modules and in Organization II, individual request buffers exist for each module.

The center of the control in the memory system is the intelligent scheduler. The scheduler, using a scheduling algorithm, decides at the beginning of each memory sub-cycle whether to initiate a memory module and, if so, which module to initiate. The selection of which module to initiate is determined by the information about the requests in the associative buffers and by the knowledge about the status of the modules (free or busy). Three scheduling algorithms are investigated in this design.

In data compression, the existing techniques have been classified into four areas: run length encoding, differencing, statistical encoding and value set schemes. A multi-level compression scheme is proposed so that data is compressed through a set of cascaded stages.

## REFERENCES

[AND75] Anderson, G. A. and Jensen, E. D., "Computer interconnection Structures: Taxonomy, Characteristics and Examples," Computing Surveys, vol. 7, no. 4, December 1975.

[BAS70] Baskett, F., Browne, J. C., and Raike, W. M., "The Management of a Multi-level Non-paged Memory System," Spring Joint Computer Conference, 1970, pp. 459-465.

[BOL67] Boland, L. J., Granito, G. D., Marcotte, A. V., Messina, B.V. and Smith, J. W., "The IBM System 1360 Model 91: Storage Systems," IBM Jour. of Res. and Dev., January 1967, pp. 54-68.

[BRI77] Briggs, F. A. and Davidson, E. S., "Organization of Semi-conductor Memories for Parallel - Pipelined Processors," IEEE Trans. on Comp., vol. C-26, no. 2, February 1977, pp. 162-169.

[BUR70] Burnett, G. J., and Coffman, Jr., C. G., "A Study of Inter-leaved Memory Systems," Proc. AFIPS 1970 SJCC, vol. 36, pp. 467-474, AFIPS Press, Montvale, N.J.

[BUR73] Burnett, G. J. and Coffman, Jr., E. G., "A Combinational Problem Related to Interleaved Memory Systems," JACM, vol. 20 no. 1, January 1973, pp. 29-45.

[BUR75] Burnett, G. J. and Coffman, Jr., E. G., "Analysis of Inter-leaved Memory Systems Using Blockage Buffers," CACM, vol. 18, no. 2, February 1975, pp. 91-95.

[CHA77] Chang, D. Y., Kuck, D. J., and Lawrie, D. H., "On the Effective Bandwidth of Parallel Memories," IEEE Trans. on Comp., May 1977, pp. 480-490.

[DRA66] Draper, N. R. and Smith, H., Applied Regression Analysis, John Wiley and Sons, New York, 1966.

[FAJ73] Fajman, R., and Borgelt, J., "WYLBUR: An interactive text editing and remote job entry system," CACM, vol. 15, no. 5, May 1973, pp. 314-333.

[FLO64] Flores, I., "Derivation of a Waiting-Time Factor for a Multiple Bank Memory," JACM, vol. 11, no. 3, July 1964, pp. 265-282.

[FOS68] Foster, C. C., "Determination of priority in associative memories," IEEE Trans. Electron. Comput., vol. EC-17, August 1968, pp. 768-789.

[HEL67] Hellerman, H., Digital System Principles, McGraw Hill, New York, 1967, pp. 228-229.

[HOO77] Hoogendoorn, C. H., "A General Model for Memory Interference in Multi-processors," IEEE Trans. on Comp., vol. C-26, no. 10, October 1977, pp. 998-1005.

[HUA74] Huang, T., "An upper bound on the entropy of run length coding," IEEE Trans. on Info. Theory, vol. IT-20, no. 9, September 1974, pp. 675-676.

[KNU73] Knuth, D. E., The Art of Computer Programming. Vol. 3: Sorting and Searching. Addison-Wesley, Reading, Mass., 1973.

[KNU75] Knuth, D. E. and Rao, G. S., "Activity in an Interleaved Memory," IEEE Trans. on Comp., vol. C-24, no. 9, September 1975, pp. 943-944.

[NUT77] Nutt, G. J., "Memory and Bus Conflict in an Array Processor," IEEE Trans. on Comp., vol. C-26, no. 6, June 1977, pp. 514-521.

[RAM78a] Ramamoorthy, C. V., Turner, J. C. and Wah, B. W., "A Design of a Cellular Associative Memory for Ordered Retrieval," IEEE Trans. on Comp., vol. C-27, no. 9, September 1978.

[RAV72] Ravi, C. V., "On the Bandwidth and Interference in Interleaved Memory Systems," IEEE Trans. on Comp., vol. C-21, no. 8, Short Notes, August 1972, pp. 899-901.

[SAS75] Sastry, K. V., and Kain, R. Y., "On the Performance of Certain Multiprocessor Computer Organizations," IEEE Trans. on Comp., vol. C-24, November 1975, pp. 1066-1074.

[SCH71] Schieber, W. D., and Thomas, G. W., "An Algorithm for Compaction of Data," J. Library Automation, vol. 1, no. 4, pp. 198-206.

[SKI69] Skinner, C. E., and Asher, J. R., "Effects of Storage Contention on System Performance," IBM Sys. J., no. 4, 1969, pp. 319-333.

[SMI77] Smith, A. J., "Multi-processor Memory Organization and Memory Interference," CACM, vol. 20, no. 10, October 1977, pp. 754-761.

[STR70] Strecker, W. D., Analysis of the Instruction Execution Rate in Certain Computer Structures, PhD Thesis, Carnegie Mellon University, Pittsburgh, Pa., 1970.

[TOD76] Todd, S. J. P., "The Peterlee Relational Test Vehicle - A System Overview," IBM Sys. J., vol. 15, no. 4, December 1976, pp. 285-308.

[TOM67] Tomasulo, R. M., "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," IBM J. of Res. and Dev., January 1967, pp/ 25-33.